1993-09

A genetic algorithm based anti-submarine warfare simulator

Timmerman, Michael Jay

Monterey, California. Naval Postgraduate School

Approved for public release; distribution is unlimited

A GENETIC ALGORITHM BASED ANTI-SUBMARINE WARFARE SIMULATOR

by

Michael Jay Timmerman
Lieutenant Commander, United States Navy Reserve
B.S., United States Naval Academy, 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Dr. Ted Lewis, Chairman,
Department of Computer Science

REPORT DOCUMENTATION PAGE

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1993	Master's Thesis, 07/91 to 09/93

4. TITLE AND SUBTITLE
A GENETIC ALGORITHM BASED ANTI-SUBMARINE WARFARE SIMULATOR

5. FUNDING NUMBERS

6. AUTHOR(S)
Timmerman, Michael Jay

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5118

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research was aimed at improving the genetic algorithm used in an earlier anti-submarine warfare simulator. The problem with the earlier work was that it focused on the development of the environmental model, and did not optimize the genetic algorithm which drives the submarine. The improvements to the algorithm centered on finding the optimal combination of mutation rate, inversion rate, crossover rate, number of generations per turn, population size, and grading criteria.

The earlier simulator, which was written in FORTRAN-77, was recoded in Ada. The genetic algorithm was tested by the execution of several thousand runs of the simulation, varying the parameters to determine the optimal solution. Once the best combination was found, it was further tested by having officers with anti-submarine warfare experience run the simulation in various scenarios to test its performance.

The optimum parameters were found to be: population size of eight, five generations per turn, mutation rate of 0.001, inversion rate of 0.25, crossover rate of 0.65, grading criteria of sum of the fitness values of all alleles while building the strings, and checking the performance against the last five environments for the final string selection. The use of these parameters provided for the best overall performance of the submarine in a variety of tactical situations. The submarine was able to close the target and execute an attack in 73.1% of the two hundred tests of the final configuration of the genetic algorithm.

14. SUBJECT TERMS
Genetic algorithm, anti-submarine warfare, simulator

15. NUMBER OF PAGES
111

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

ABSTRACT

This research was aimed at improving the genetic algorithm used in an earlier anti-submarine warfare simulator. The problem with the earlier work was that it focused on the development of the environmental model, and did not optimize the genetic algorithm which drives the submarine. The improvements to the algorithm centered on finding the optimal combination of mutation rate, inversion rate, crossover rate, number of generations per turn, population size, and grading criteria.

The earlier simulator, which was written in FORTRAN-77, was recoded in Ada. The genetic algorithm was tested by the execution of several thousand runs of the simulation, varying the parameters to determine the optimal solution. Once the best combination was found, it was further tested by having officers with anti-submarine warfare experience run the simulation in various scenarios to test its performance.

The optimum parameters were found to be: population size of eight, five generations per turn, mutation rate of 0.001, inversion rate of 0.25, crossover rate of 0.65, grading criteria of sum of the fitness values of all alleles while building the strings, and checking the performance against the last five environments for the final string selection. The use of these parameters provided for the best overall performance of the submarine in a variety of tactical situations. The submarine was able to close the target and execute an attack in 73.1% of the two hundred tests of the final configuration of the genetic algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my wife Shereen for her wonderful support during my two years at Naval Postgraduate School, and Dr. Shing for the expertise, patience, and time he graciously provided me.

I. INTRODUCTION

A. STATEMENT OF PROBLEM

Artificial intelligence holds great promise as a tool to be used in tactical simulators. Genetic algorithms seem particularly well-suited to this task. They have the ability to sort through the many choices/options available in a tactical scenario, provide an "intelligent" solution, and yet retain an element of uncertainty in the response to a particular scenario. The use of a genetic algorithm as the "mind" behind the computer model of a submarine fits this mold. The anti-submarine warfare environment is often vague, with inexact sensors providing the players with inexact information. This is combined with the great number of possibilities (various courses, speeds, depths, attack methods, et cetera) available to the submarine/computer model.

Providing for the optimal genetic algorithm, however, is not so simple. There are many factors to be considered, once the basic tactical scenario has been decided upon. The tactical scenario dictates the makeup of the members of the population used in the algorithm (in this case, the changes in course, speed, depth, and attack choices, and the sensor data available to the modeled submarine.) However, the specifics of how the algorithm manipulates these members of the population, to consistently provide an optimal, or near-optimal, solution to the scenario, remain to be decided. These include the size of the population, the method of evaluating the members of the population for their relative strengths, the method of choosing members for the new generation and choosing the mating pairs within a generation, the rates of successful mating, and probabilities of inversion and mutation.

The goal of the agent/model in this case is to successfully attack the target, while avoiding being attacked. This takes place in a constantly changing "environment" due to updates on the target's position, speed, relative motion, etc. While the submarine is stalking the target, the target has the goal of attacking the submarine, so a direct, high-speed

approach by the submarine is probably not the optimal solution: it may accomplish the partial goal of reaching the target, but this may be at too high a cost, if it is itself sunk in the process. These goals can be almost mutually exclusive, hence the difficulties in determining the optimal algorithm.

B. APPROACH

The purpose of this research is to optimize the genetic algorithm used in earlier research by Hayden [HAY91] to model a submarine. This will include the translation of the earlier work from FORTRAN 77 into Ada. This research will test the areas mentioned earlier, combining the many portions of the algorithm into a general, optimal solution for this model, as well as making some improvements in the earlier model.

Using Hayden's research as the basis, the research will examine various versions of the model, determining those which have the best performance, in different tactical situations. Based on the results of the tests, the optimal model for the scenario will be chosen.

C. SUMMARY OF CHAPTERS

Chapter II discusses the background of the algorithmic theory and the earlier work on this model in more detail. Chapter III deals with the specific genetic algorithm used in this model. Chapter IV describes the specific program of tests designed and run to determine which version of the model worked best. Chapter V provides the summary and conclusions. Appendix A shows the tactical decisions made by the submarine and ship, and graphically displays the moves from two ship/submarine encounters using the model. Appendix B contains instructions for using the simulator. Appendix C shows in more detail the payoff values referred to in Chapter IV. Appendix D contains the Ada code for running the model.

II. PREVIOUS WORK

A. BACKGROUND

The earlier research on this model was initiated in response to the need in the United States Navy for effective, efficient, inexpensive anti-submarine warfare (ASW) simulators [HAY91]. Although there has been a significant lessening of the submarine threat in recent years, it still exists. This is especially true in light of the increasing sales of sophisticated weapons systems, including submarines, to third world countries. (E.g. the recent acquisition of submarines by Iran.)

With significant cutbacks in the Navy's budget already, and further cuts on the horizon, it is vital that the most is made out of the available training dollars, range time, and submarine/target services. To accomplish this, it is important that the more basic aspects of training be completed in a less-expensive, more convenient manner. A model such as this provides the user the opportunity to learn/rehearse/refine basic ASW skills in an inexpensive, convenient, user-friendly environment. This provides a more skilled user to the more sophisticated, expensive, and less convenient simulator/training periods, which results in better utilization of these opportunities.

B. EARLY RESEARCH IN MACHINE LEARNING

There are many types of artificial intelligence models, each with its own set of advantages and disadvantages. Expert systems use sets of conditional (if-then) statements, provided by a human expert, to show behavior similar to that which humans would display were they making the decisions. The expert system applies these statements, or production rules, to the initial data provided, and determines the appropriate response. This has the result of always providing the same response to the same data set. In many situations this is an advantage, but when attempting to model a submarine's tactics this has the disadvantage of causing predictability. Anti-submarine warfare takes place in an imperfect

environment, where sensors often do not provide complete or exact information, and the outside (water) conditions are constantly changing. Both submarine and surface ship are working against an opponent commanded by a human, who may or may not act according to "the book." In such a world, complete predictability is not an advantage for a training model.

Although there are basic "rules" normally followed in a given tactical situation, much depends on what one commander "feels" his opponent may do next. Thus, to effectively simulate this, the model must be able to "learn" from previous experiences, keeping what has worked, and removing what hasn't from its store of knowledge/production rules. This has similarities to the selection process found in nature, where the poor performers tend to be discarded and the stronger tend to survive.

Rosenblatt's Perceptron was a model which used a process based on the stimulus-response framework in the retina of the human eye [YAZ86, p. 212]. Sensory units passed inputs on to the memory. The memory units were acted on by the synapses, and the result was output through the response units. The response units also provided feedback to the synapses. The synapses provided the weighted decision-making on the data contained in the memory units. A limit to this model was its inability to reach a conclusion/response that was outside the realm of the input signals [FOG86, p. 12].

The evolutionary programming concept used in Fogel's machine-learning model was based on the natural order, with mutation and survival of the fittest. The machine's fitness is based on it's logic structure in relation to previous environments. Thus the strongest machine has a logic which best matches the previous environment. The model attempts to predict what the next environment/input will be. The internal states of the machine are probabilistically mutated (by varying the numbers of states, outputs, transitions between states, etc.). The resulting machine is compared, along with the parent machine, to the next input. Those who fare better than the parent are kept, those who don't are discarded.

This process continues until the end result (i.e. a particular score) or some limit (i.e. processing time) is reached. Fogel's model allowed for an expansion (through the mutations) of the input alphabet, improving the ability to "learn" and advance [FOG86, p. 29].

Machine learning has been defined as improvement in a computer system's performance over time without being reprogrammed. To be effective and efficient, this must be accomplished over a wide range of problems/inputs, through the updating of the production rules (the portion of the model which operates on the inputs). But what is improvement defined as, and how should it be measured? There needs to be a method of measuring intermediate goals, because to rate a machine/algorithm purely on reaching an end goal would be too much of a hit or miss proposition.

John Holland was the first to use a system which was modeled after the adaptation seen in a natural environment. This treatment is called genetic algorithms. Genetic algorithms consist of a set of operations which are performed on the "population" of strings [GOL89, pp. 62-68, 166]. The strings are made up of a set of individual elements, called "alleles," which contain independent rules. The operations which are performed on the strings are designed to create a stronger population, better suited to performing the desired task(s) than its predecessor. These operations typically include selection for reproduction, mating of the strings chosen for reproduction, mutation of a small percentage of the individual elements of the strings, and inversion of the elements of an individual string. There are numerous ways each of these operations can be carried out, each with its own advantages and disadvantages.

C. PREVIOUS USE OF A GENETIC ALGORITHM IN AN ASW SIMULATOR

Hayden's wargame models a Victor class submarine, run by the program, against a Knox class frigate, towing an SQR-18A towed array sonar. The primary sensor of both

submarine and ship is passive sonar, which presents inexact and sometimes confusing data to its users. The scenario is centered on an area in the North Atlantic east of the Gulf Stream. The environmental data provided for the simulation accurately reflects the conditions in this locale.

There are six major modules in the program: initialization, detection, ship decision and attack, submarine decision and attack, movement, and administration (Figure 1). The

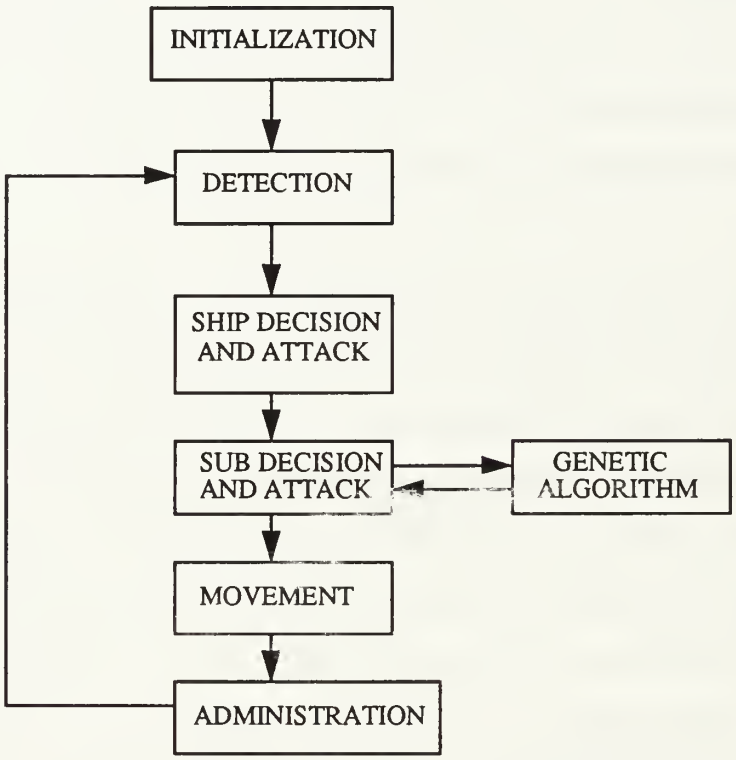

Figure 1. Flow Diagram of Main Modules in Model

initialization phase reads in the environmental data and sets up the initial conditions and parameters. It calculates the move state space, environmental state space, and joint state space values, and, finally, randomly fills the strings for the initial generation of the population. The detection module, using the basic sonar equation, determines whether the

ship detects the submarine and vice versa. Using random numbers and probabilities, it determines whether the ship and submarine detect each other with radar, visually or with ESM gear if the submarine is at periscope depth. In the ship decision and attack module, the user makes desired changes to his current tactics. If the ship attempts to attack, the attack will be evaluated based on the entry position of the ship's torpedo and the position of the submarine. The submarine decision and attack module contains the genetic algorithm portion of the program. If the submarine holds contact on the ship, the genetic algorithm determines what the next set of tactics will be. When not in contact, a lost contact or random search procedure will be used, whichever is appropriate at the time. If the algorithm chooses to attack, and the other parameters (i.e. range, detection status) are appropriate, the attack will be conducted, and evaluated for success or failure. If the range criteria are met, the success or failure of the attack is decided by random number selection.

The movement module updates the positions of the ship and submarine each turn and advances the game clock. Finally, the administration module keeps a library of all pertinent tactical data from the wargame, and provides recap information (i.e. courses, speeds, environments, tactical decisions for each turn) at the conclusion of the simulation.

III. THE GENETIC ALGORITHM

A. OVERVIEW

This chapter will present the concepts used to model the submarine and its environment, and discuss the basic theory behind the decisions made by the submarine in its efforts to reach its goal. The basis of the genetic algorithm is that the model/submarine will periodically evaluate its progress towards its goal, and use this evaluation to update its decisions in subsequent situations. In our model, the strings are made up of 512 elements. These correspond to the 512 possible environments based on the following inputs to the submarine's sensors: contact type, range estimate, doppler, bearing drift, bearing trend, bearing rate, contact strength. Each element contains a value between 1 and 504, inclusive, indicating which set of the 504 possible tactical combinations it holds. The 504 combinations represent the models possible course, speed, depth and attack combinations. These choices are shown in more detail in Tables 1 and 2. Since each environment and each combination of tactics have an associated numerical value, they will be combined into a joint space state value discussed later.

B. VALUATED STATE SPACE

As stated earlier, the goal of the submarine in this model is to successfully attack the surface ship, while avoiding its own destruction. But what is the best way to measure the intermediate goals, which allow the submarine to reach this final destination? There must be some system of reward to encourage behavior that approaches the end goal, and discourages behavior that causes the distance to the end goal to increase.

Rather than resorting to an expert system, with specific paths towards the end goal for each situation, a more realistic answer (incorporating some of the uncertainty and human element always present in the inexact world of antisubmarine warfare) is to use a valuated state system which achieves the above-mentioned target of rewarding behavior that

approaches the goal. This will assign point values to the various actions the submarine can take. Our model has four possible actions: to attack or not to attack, to change course, speed, or depth. These will each be weighted according to the expected effect on nearing the end goal: e.g. choosing the "attack" option will be more highly rewarded than choosing the "not attacking" option. The various options open to the submarine model are shown in Table 1. As shown by the point values, those actions which are more likely to further the submarine towards its goal have higher point values than actions which decrease the likelihood of successfully reaching the target. Each of the four tactical choices has a weighted value representing the relative importance of that particular choice towards realization of the end goal(s). The substate weight is the relative value of that particular substate. This will be multiplied by the "Action Worth" to determine the total point value for a particular tactic, and each of the four tactics will have their values summed up for the overall value of a particular set of tactics. (See Equation 1 on page 12.) However, a point system based on the actions of the submarine alone does not describe adequately the progress towards the eventual goal. If the target is steaming away from the submarine at a speed of ten knots, it doesn't matter if the submarine has selected the attack option, and is headed for the target at a speed of nine knots. The distance between them will continue to increase. This is where an evaluation of the environment comes into play.

In this instance, the environment, as seen by the submarine, consists of it's perception of the target, as viewed through the inexact sensors at its disposal. (This model operates primarily with passive sonar, not the more accurate active sonar and radar, which can also pose a threat to the user through counterdetection.) The seven parts of the environment are: type of contact, range to contact, doppler, bearing drift, bearing trend, bearing rate, and contact strength. These will be assigned values opposite in direction to those of the submarine's actions: no contact has a higher rating than does strong contact. The seven environmental inputs, with their associated values, are shown in Table 2. There are 512 different combinations of the environment the submarine might find itself in.

TABLE 1: SUBMARINE VALUATED STATE SPACE

Submarine's Goal		Destroy the Surface Combatant	
Substate	Substate Weight	Action	Action Worth
Attack	10	attack	9
		do not attack	0
Course Offset (degrees)	10	45	15
		30	14
		60	10
		0	8
		90	5
		-90	4
		180	3
Speed (kts)	7	8	10
		12	7
		16	4
		4	4
		20	2
		24	1
Depth (ft)	3	no change	9
		60	8
		150	8
		300	6
		450	2
		600	1

TABLE 2: ENVIRONMENT VALUATED STATE SPACE

Ship's Goal		Evade The Submarine	
Substate	Substate Weight	Action	Action Worth
Contact	10	none	10
		broadband	4
		narrowband	2
		broadband & narrowband	0
Range	10	unknown	10
		far	9
		middistance	4
		near	0
Doppler	8	down	3
		up	1
Bearing Drift	7	not steady	2
		steady	1
Bearing Trend	5	not steady	1
		steady	0
Bearing Rate	7	not steady	2
		steady	1
Contact Strength	2	weak	1
		strong	0

Thus, at any given moment, there is a total point value associated with the submarine's perception of the target (the environment), and a separate value associated with the combination of tactics the submarine has chosen. When these two are combined, by determining the difference between the two, the joint state space value has been

determined. Once the valuated state space has been constructed its current value is computed using a normalizing function. Let the value of the present state to X be $V_x(S_o)$, the value of the current set of the i^{th} substate be VH_{i_o}, and the weight of the i^{th} substate be WR_i. The normalizing function is shown in Equation 1.

$$V_x(S_o) = \frac{\sum_{i=1} \left(\frac{(VH_{i_o})}{max\,(VH_i)} \right) WR_i}{\sum_{i=1} WR_i} \qquad \text{(Eq 1)}$$

For example, a submarine at the state consisting of (attack, offset course by 45 degrees, new speed of 12 knots, maintaining current depth) will have a state value of:

$$\left(\frac{\frac{9}{9} \times 10 + \frac{15}{15} \times 10 + \frac{7}{10} \times 7 + \frac{9}{9} \times 3}{10 + 10 + 7 + 3} \right) = 0.93$$

Likewise, an environment consisting of (broadband contact, near range, down doppler, steady bearing drift, steady bearing trend, steady bearing rate, weak contact strength) will have a state value of:

$$\left(\frac{\frac{4}{10} \times 10 + \frac{0}{10} \times 10 + \frac{3}{3} \times 8 + \frac{1}{2} \times 7 + \frac{0}{1} \times 5 + \frac{1}{2} \times 7 + \frac{0}{1} \times 2}{10 + 10 + 8 + 7 + 5 + 7 + 2} \right) = 0.39$$

The valuated state space uses the joint state space to define the interaction between the opponents. This is evaluated as shown in Equation 2. The current set of tactics is combined with the current set of environmental values for the joint state space value.

$$V_j(S) = (V_x(S) - V_y(S)) \qquad \text{(Eq 2)}$$

For example, the joint state of the above environment and tactical state will have a joint state value of: (0.93 - 0.39= 0.54). The greater the value, the better that set of tactics/ environments is for unit *X*; the smaller the value, the better that set is for unit *Y* [HAY91].

C. ELEMENTS OF THE GENETIC ALGORITHM

The operations on the population of strings in the algorithm include mutation, inversion, and crossover. In addition, the size of the string population, the manner in which the members are chosen for reproduction, the method of evaluating the strength of the strings, and the selection process for mates play important roles in the algorithm.

1. Population Size

The size of the population affects the performance of the algorithm in several ways. Larger populations achieve a greater sampling of the search space, at least with the initial (random) generation of the allele strings [SCH86]. The larger the population, the less likely is premature convergence, (i.e. the population converges on a local optimal solution too early). However, if the population is too large, there may be an unacceptably slow rate of convergence [GRE86] as a larger population has a greater "fitness inertia" [TAT93]. This results in the algorithm taking longer than desired to achieve the optimal solution. Past experiments have found that population size of 30-100 provides the best on-line performance [GRE86]. The algorithm presented in this thesis uses a static population size.

2. Evaluation Method

There are a number of methods of evaluating the intermediate (en route to the ultimate goal) performance of the algorithm. The two chosen for examination here are as follows: 1) Rating each string on its total strength by summing up the environment/tactical differences (joint space state value) of each allele of the string, and choosing the string with

the highest overall value. 2) Taking the performance of each string against the last five environments, totaling the values, and choosing the string with the highest total value. The string that is chosen is the one used to determine the move (tactical options) in the current environment. (This occurs after the strings have had the genetic operations performed on them.)

3. Selection/Choosing Offspring

The evaluation function mentioned above is also used in determining which strings are selected for reproduction. Once the strings in a generation have been evaluated and ordered by their evaluation rankings, a random number generator is used to determine which strings will be chosen for the new generation, based on string's percentage of the total value of all strings in the population. This results in, on the average, the strings being reproduced in proportion to their relative strength.

4. Choosing Mating Pairs

Once the strings for the next generation have been chosen, they are paired off for the mating (crossover) operation. The new strings are sorted by the evaluation function, and then mates are chosen, with a primary goal of avoiding "incest" (the mating of one string with a duplicate/very similar string.) This helps to avoid the number of duplicates, which in turn limits the potential for premature convergence. Previous studies [ESH91] have shown that incest prevention is always a good idea. Of course, if more than half of the population are duplicates/near duplicates, some mating of duplicates will occur.

5. Crossover

Crossover avoids the problem created when selection is the only operation performed, that of a new generation cloned from the higher performers of the parent generation. Combined with selection, crossover creates new strategies. The role of crossover is "...to introduce diversity into the population probing new regions unexplored by the selection operator" [TAT93]. The crossover operation combines the values of two

members of the population by the exchange of certain allele values between the two strings. The pair is selected as discussed earlier, and the position for the value exchanges is chosen randomly. Crossover, while producing new strings, uses only the information currently available in the population. If the rate of crossover is too high, high-performance strings will be altered before they can produce improvements. If the rate is too low, the search for the optimal strategy will proceed too slowly [GRE86].

6. Mutation

In each generation, a certain, small number of the alleles of the population are randomly selected to be altered by the process of mutation. A member of the "alphabet" (possible values for the allele) is randomly selected to replace the former value held by the selected allele. This introduces new material into the population. The new material does not come from the parents, and is not the result of the crossover or inversion operations. (Inversion will be described in the next section.) It assures that the entire search space is connected [GEN87][KUO 93]. Without mutation, the population would be limited to the possibilities present in the initial population, a very small percentage of the total available.

However, mutation must be used with care. An adequate level of mutation will prevent alleles from converging too soon on a particular value, and inferior values/solutions can be experimented with by the algorithm without having a severe impact on the overall performance [SCH86]. If the level of mutation is too high, though, the result will be a search which approaches randomness in its strategy. Suggested mutation rates are in the range of 0.001-0.01.

7. Inversion

Inversion is a form of mutation where certain alleles within a particular string exchange values. While not providing the pure randomness of the mutation operation, it also prevents individual alleles from converging on a single value. However, inversion does not explore the members of the alphabet which are not contained within a certain string, as does mutation.

8. Preventing Premature Convergence

When the population has converged somewhat on a single solution (string of alleles), the genetic algorithm will continue its search around the neighborhood of that solution. At this point, mutation and inversion allow the search to continue for a more optimal solution. This search will continue until the program is halted.

One of the goals of a genetic algorithm is to have this convergence occur at the proper time. If the algorithm is designed poorly, convergence will never occur, and the optimal solution will never be found. On the other hand, if the population is allowed to converge too soon, it may converge on a local, rather than global, optimum, thus preventing the best solution from being found. This is likely to occur when members of the next population are chosen based on the relative strengths of the strings in the parent generation, and mating and mutation are not part of the algorithm. Eventually, probably within a few generations, the population will consist mostly of a few "super genotypes", which have a strength higher than the strings they started with, but not very high relative to the optimum value [WHI89].

D. FLOW OF THE GENETIC ALGORITHM

The flow of the genetic algorithm is shown in Figure 2. Figure 3 shows the interaction of the environment with the algorithm, the control system, and the performance measures. The bit strings, either those randomly selected in the initialization phase or those from the previous turn, are first graded for their relative strength. The nucleus of the next generation is then randomly selected, with the stronger strings having a higher probability of selection. The selected strings are then sorted by fitness value. Based on this sorting, each is paired with another string for the crossover operation, ensuring that strings do not crossover with identical strings. The pairs exchange allele values at certain positions, after a random number determines if and where the crossover between an individual pair will take place. The strings are then inverted and mutated, with the allele value of a certain number of positions (the positions determined randomly) mutated, to a random member of the

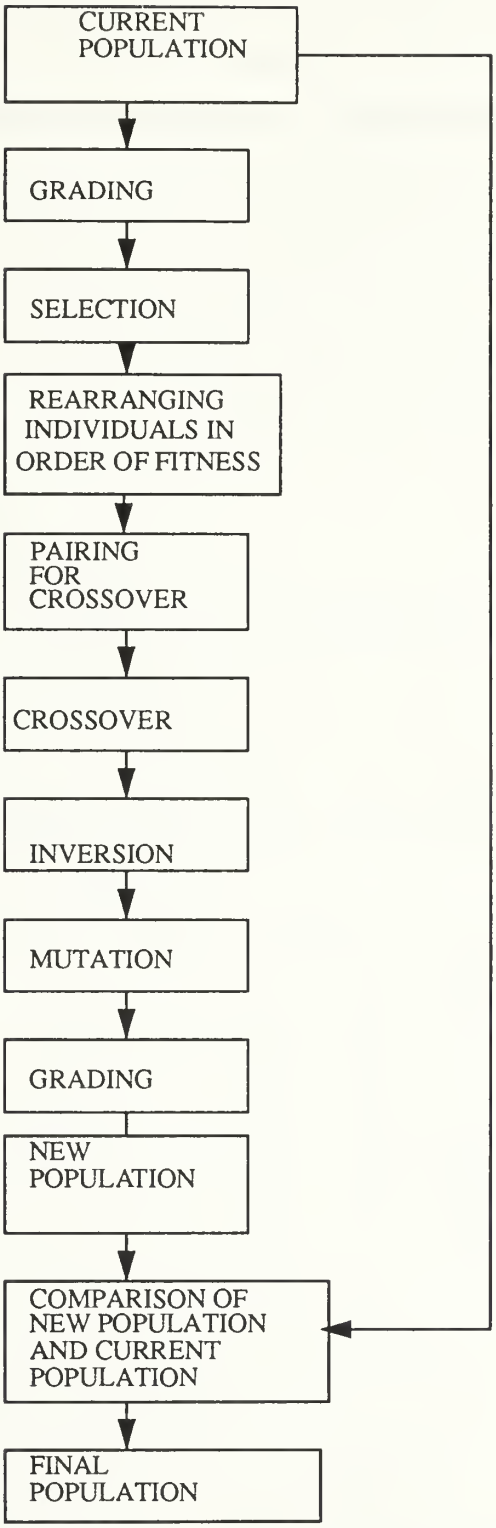

Figure 2. Flow Diagram of the Genetic Algorithm

alphabet. The strings are then graded again, and the final copy of the "new" generation is compared with the "old" generation. The generation with the highest overall strength is selected. This procedure is repeated a designated number of times each turn, and the final generation obtained at the end of the evolution is returned to the wargame. The next set of submarine tactics is determined by picking the appropriate (based on the current environment) allele's value from the strongest string in the population. A more detailed description of the individual procedures follows. Data on test cases used to determine specific parameters is included in Chapter IV.

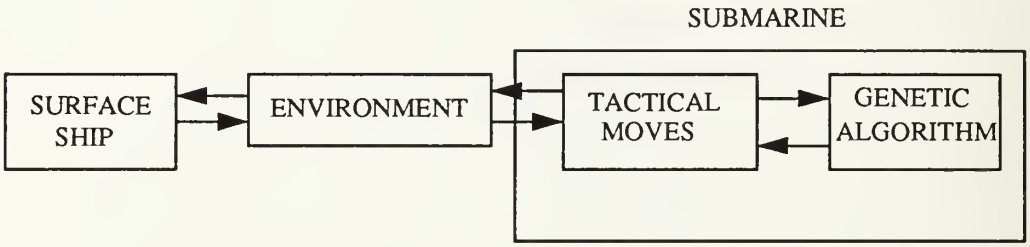

Figure 3. Relationship of Algorithm, Environment, and Tactics

Two versions of a grading procedure were tested. The first checked each of its 512 alleles against its corresponding environment for the joint state space value, totaled the values, and ranked the strings in order of the sum. The second version only checked the five alleles corresponding to the five most recent environments, summed up the five joint state space values, and ranked the strings in order of the greatest sums. (An estimate on probable environments was used for the initial five environments used.) This version was also tested with varying weights applied to the joint values, giving greater weight to the joint values from the more recent environments. The total strength of the population, and the proportion of this strength held by each string, was also computed. The final choice for the grading procedures combined these two, using the former version while the population was selected and went through the genetic operations, and the latter version for the final selection of the strongest string responsible for the submarine's actual tactical move during that turn. This proved to be the most successful.

These values are used in the selection process, where a random number is compared with the values of the strings to determine which one is chosen for inclusion in the next generation. For example, with a population of four, string A holds 38 percent of the total value, B has 27 percent, C has 20 percent, and D has 15 percent. If the random number falls between 0 and 0.38, A is chosen, between 0.38 and 0.65, B is chosen, etc. This is executed once for each member of the new population.

The strings are now paired off in such a way to avoid strings mating with identical strings. (This would result in no change when the strings executed a crossover, which leads to premature convergence.) The strings with the highest values are assigned "odd" positions and the remaining strings are assigned the remaining, "even" positions. Each string then "mates" with its neighbor. A random number checked against the crossover probability of 0.65 determines the success rate of the "mating." If this criteria is met, the crossover procedure is called. The crossover site (allele position) is determined randomly. All alleles values from this point to the end of the string are exchanged with the values in the corresponding allele of the mating pair.

Following the crossover procedure, the strings are selected for the inversion procedure. This occurs on a string-by-string basis, with probability of inversion of 0.25. Once a string has been selected, the beginning and ending alleles for the string are selected. The values held in the inclusive alleles are then switched with the allele holding the corresponding position. For instance, if positions two and five were chosen as the two endpoints, two and five would exchange values, and positions three and four would exchange values.

The mutation procedure randomly mutates a set amount (0.1 percent) of the alleles in each population. For instance, in a population of sixty-four (32768 total alleles), thirty-three alleles would have their values mutated each generation. The new value is a randomly chosen member of the "alphabet" (the set of tactical options.)

IV. TEST AND EVALUATION

The testing of the model was performed in several phases. The parameters that were examined were population size, grading procedure, crossover and mutation rates, various inversion rates (including elimination of the procedure) and varying numbers of generations for each turn of the wargame. The algorithm was tested against a ship traveling a steady course, one executing a random zig-zag pattern (centered on a constant course), a ship heading away from the submarine at slow speeds, and against a human opponent attempting to attack the submarine. The specific test runs shown in Appendix A were selected because they show certain traits worthy of further examination.

One of the difficulties involved in determining the optimum combination of parameters is the high usage of random numbers in the model. This makes it difficult, perhaps impossible, to completely compare two test runs. Random numbers are used to select the initial bit strings, to select strings for reproduction, alleles for mutation, crossover positions, success of the crossover procedure, to name some of their uses. Thus, multiple runs for each set of parameters were needed to ensure that success or failure was not the result of an unusual sequence of random numbers. The individual sets of parameters were then graded on the overall performance over the set of runs, and this data was used to determine the relative strength of the different parameter sets. Parameters were evaluated one at a time, with all other parameters held constant. For example, the probability of mutation was changed from 0.001 to 0.005, and multiple runs were performed at this setting, with population size, inversion rate, crossover rate, and grading procedure held constant. The results were compared to the run results from the old setting, and the final value for mutation rate was chosen.

The success of the model on a particular test case was determined by the following: a successful run was one in which the submarine executed a torpedo attack on the target. (The ultimate success or failure of the attack was not considered, since this is determined by a

random number versus a hit probability.) Partial success was achieved if the submarine closed to within five nautical miles of the target (an arbitrarily chosen figure) but did not execute an attack. The number of turns in which the "attack" option was selected was calculated, and finally, the percentage of turns from a case in which the submarine-to-target range decreased from one turn to the next was checked. Tactics which decreased the submarine-target range indicates that the genetic algorithm had selected a course of action which would make progress toward reaching an eventual attack solution, even if this final result did not occur. If the "attack" option was selected, this would allow the submarine to fire a torpedo if the range/contact strength criteria were met. Without this selection, no attack would be made, no matter what the submarine-to-target range was. Hence, we include "% Turns Attack Selected" and "% Turns Range Decreased" in the tables to indicate the relative effectiveness of the various parameter settings.

A. THE EFFECTS OF PAYOFF VALUES

The payoff values were changed to increase the award given for selecting tactics' combinations which result in achieving/approaching the final goal of attacking the surface ship. (Details of the payoff values can be found in Appendix C.) As shown in Table 3 there

TABLE 3: PAYOFF VALUES

Payoff	1	2	3	4
Test Cases	20	20	20	20
Successful	7	15	17	18
Partial Successes	5	1	1	1
%Turns Attack Succeeded	8.1	47.6	32.2	53.0
% Turns Range Decreased	40.3	58.9	64.6	57.7

is a significant improvement in the performance of the model with higher payoff values. There was less randomness in the submarine's tactics, and in general a better approach to the scenario.

B. THE CHOICE OF GRADING PROCEDURE

The procedure which rates the members of the population, used to choose members for reproduction, select crossover mates, and choose between generations plays a very important role in the algorithm. It was found that summing the joint state space values for each allele in each string to determine fitness produced a better result than checking the performance of each string against the last five environments. Unfortunately, because of the significantly greater number of computations involved, this resulted in a much slower run time for the model. A combination of the two, where the final selection is made by checking the last five environments, and the other gradings are done by totalling all alleles, was found to be the most successful. Table 4 shows the results.

TABLE 4: GRADING PROCEDURE

Procedure	Total of all Alleles	Last Five Environments	Combined Procedure
Test Cases	20	20	20
Successful Cases	16	12	19
Partial Success	2	1	1
% Turns Attack Selected	11.2	62.5	54.7
% Turns Range Decreased	55.4	58.3	84.0

C. POPULATION SIZE

Table 5 shows the findings when different population sizes were tried. A population size of 8 was found to have the best performance. The scenarios run for these tests were as follows: 1) 10 runs with the ship zig-zagging towards the submarine at 15 knots, 2) 10 runs with the ship zig-zagging towards the submarine at 15 knots, then slowing to five knots after eight turns, 3) 10 runs with the ship heading towards the submarine at 15 knots, then after eight turns, turning away at three knots, 4) 20 runs with the ship heading away from the submarine at three knots from initiation of the simulation.

TABLE 5: POPULATION SIZES

Population Size	4	8	16	32	64
Test Cases	50	50	50	50	50
Successful Runs	36	37	31	33	33
Partial Successes	0	1	0	0	2
%Turns Attack Selected	44.7	52.8	46.3	45.7	45.9
% Turns Range Decreased	55.5	53.7	49.8	48.7	49.6

D. MUTATION RATE

A high (one percent) mutation rate injects too much randomness into the population. A more reasonable rate (0.1 percent) allows enough randomness to optimize convergence, while not too much to greatly upset the stronger strings as they develop. (Table 6.)

TABLE 6: MUTATION RATE

Probability of Mutation	.001	.005	.010
Test Cases	10	10	10
Successful Cases	9	4	3
Partial Successes	0	3	1
% Turns Attack Selected	16.7	16.7	8.6
%Turns Range Decreased	76.9	47.4	45.3

E. NUMBER OF GENERATIONS BETWEEN PLAYS

Each turn the genetic algorithm is executed a certain number of times before the final population is returned to the model. Too many generations per turn will result in premature convergence, since the algorithm is not been able to measure the success of its tactics before the strings have converged. Not enough generations per turn will keep the algorithm from performing the genetic operations enough to optimize the population. Five generations per turn proved to have the best performance. (Table 7.)

TABLE 7: NUMBER OF GENERATIONS

Generations per turn	1	5	10
Test Cases	10	10	10
Successful Cases	3	5	1
Partial Successes	1	3	3
% Turns Attack Selected	8.6	4.4	4.4
%Turns Range Decreased	45.3	50.3	39.5

F. CROSSOVER RATE

The optimum crossover rate, the rate at which a pair of strings successfully reproduces, was found to be sixty-five percent. (Table 8.)

TABLE 8: CROSSOVER RATES

Probability of Crossover	0.50	0.65	0.80
Test Cases Cases	10	10	10
Successful Cases	6	9	5
Partial Successes	3	0	4
% Turns Attack Selected	10.3	16.7	10.9
%Turns Range Decreased	51.2	76.9	46.3

G. INVERSION RATE

The use of the inversion procedure, which injects intra-string randomness into the population, was found to be best at twenty-five percent. (Table 9.)

TABLE 9: INVERSION RATE

Inversion Rate	0.0	0.25	0.50
Test Cases	15	15	15
Successful	6	7	6
Partial Successes	3	5	5
% Turns Attack Selected	6.0	5.2	5.8
%Turns Range Decreased	40.0	57.1	42.7

H. THE TURING TEST

The genetic algorithm used the optimum parameters discussed earlier: population size of eight, inversion rate of 0.25, crossover rate of 0.65, mutation rate of 0.001, and the combined grading procedure. The submarine successfully approached the ship in all ten of these simulations, but fired the first shot in only four of them. Appendix A shows the moves made by the submarine and surface ship (played by an officer with anti-submarine warfare experience) in two simulated encounters.

In the first simulation, the surface ship was headed towards the submarine, providing the movement to decrease the range between the two. Thus, the submarine was not forced to head in the direction of the ship to achieve its goal; this occurred even with moves that might not seem to make sense. (For instance, in turns 11-15, the submarine headed perpendicular to the direction towards the ship, at a speed of 24 knots, yet was still able to progress towards its goal due to the ship's movement.) As can be seen from the ATTACK

row, the submarine's tactics do not always contain the attack directive, even though this is necessary to reach the final goal of attacking the ship. (The attack directive by itself does not indicate an attack is being made; the range and contact criteria must also be met.) The ship was able to fire the first shot in this encounter.

In the second simulation, after contact had been established by the submarine, the surface ship slowed to a speed of five knots. This forced the submarine to make different choices than in the earlier scenario. The courses chosen by the submarine have a much higher frequency of heading towards the surface ship, since the submarine was now required to provide its own momentum to reach its goal. The submarine was successful in this, and was able to attack the surface ship.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The genetic algorithm effectively simulates the operation of a submarine by generally providing realistic responses in the tactical situations it was placed in. The tactics of the submarine changed in response to the tactical environment, and showed a high rate of success in attacking the targets. However, it is difficult to optimize the algorithm for all types of situations: one that is optimal for an intercept scenario will not be appropriate when the best solution is for the submarine to hold its position and let the target come to it. An algorithm designed to chase down a target may cause the submarine to approach too quickly in one of the above scenarios. Furthermore, as observed in the experiments, the genetic algorithm caused the submarine to make many poor tactical decisions due to its random choices. This is where a genetic algorithm does not compare well to a human-in the ability to make "drastic" changes in the tactics applied to a situation. A human submarine skipper would be able to "shift gears" as required based on the current situation. Perhaps a combination of expert systems for high-level strategies and genetic algorithm generated tactics for low-level moves will result in a more realistic and effective simulator.

The best parameters for the genetic algorithm are as follows: mutation rate: 0.001, inversion rate: 0.25, crossover rate: 0.65, population size: 8. The optimal grading procedure was found to be a combination of totaling the fitness of each allele (used while changing the strings in the population) and checking for performance against the last five environments (used to pick the best string from the population for use in the next turn). The submarine was able to achieve a high rate of success, i.e. approaching and eventually attacking the target, with the algorithm using the above parameters. The relative performance of the population sizes was unexpected. The smaller population size allows for more rapid convergence to local optima, which in this environment provides a solution

adequate to the task at hand: any number of tactical solutions could get the submarine in position to attack the ship-it does not need to be the best one.

In the simulation environment, with a limited number of reasonable tactical options available, a relatively simple simulator such as this would be just as well represented by a more conventional, "if-then" based expert system. However, as complexity is added, in the form of more ships/aircraft/submarines, or as more tactical choices (missiles and torpedoes, noisemakers, tactical maneuvers such as spiralling turns) are made available, the ability a genetic algorithm has to search a large space (number of options) would be of more benefit.

B. RECOMMENDATIONS

The following areas are recommended for further research.

1. Expansion of the model to a more realistic, complex version.

The "real-world" of anti-submarine warfare is much more complex than that portrayed by this model. There are numerous types of weapons, evasion tactics, aircraft, multiple adversaries, and changing environmental conditions, to name just a few of the variables that might be included. To provide realistic training and analysis, a model must take these into account. This would greatly increase the search space used by the genetic algorithm.

2. Development of a user-friendly, desk-top trainer.

To be of maximum benefit to the Navy, a model would need to be able to be operated on a personal computer. The addition of a graphical, user-friendly display would enhance the performance as a trainer. It may be difficult to achieve this while still expanding the model as described above. A combination of more conventional expert system techniques with the genetic algorithm may help to decrease the search space required.

APPENDIX A

The tables list the data associated with the simulation shown graphically as the geographic plots of the ship and submarine. The following explanations apply to the row titles:

Contact:- "YES" if the submarine holds contact on the ship with its sonar, visually, or with radar.

Offset-Based on the bearing of the ship from the submarine. When the offset is added to this bearing, the result is the new course of the submarine. An offset of "0" would result in a course directly towards the ship, while an offset of "180" would result in a course directly away from the ship.

Speed-The speed of the submarine.

Attack-Indicates if the genetic algorithm has chosen the attack tactic.

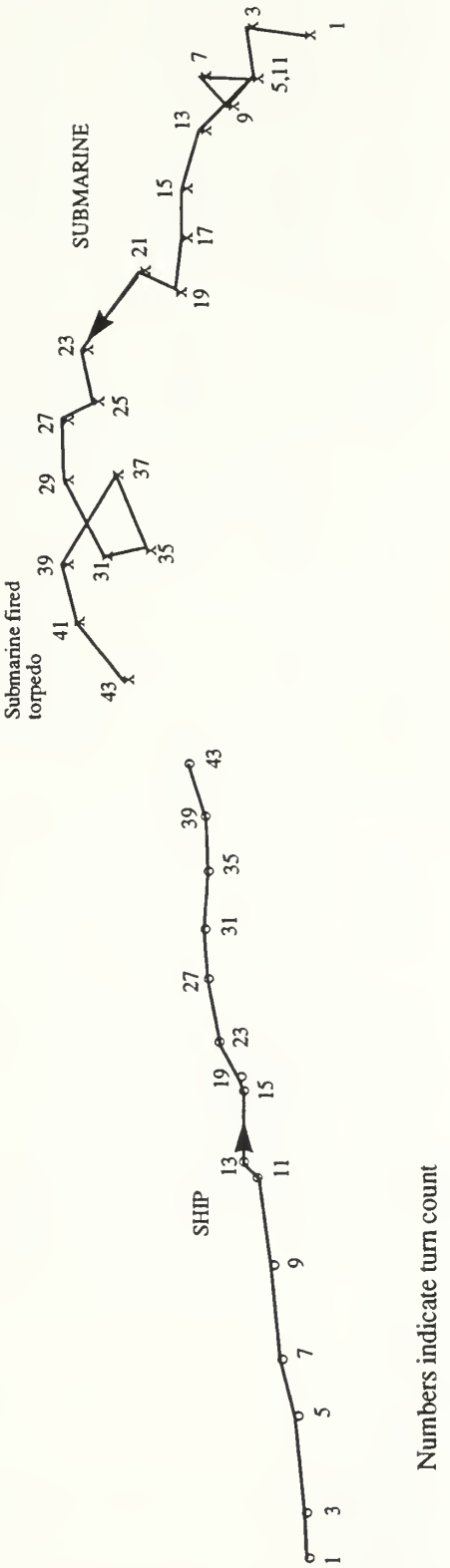

Figure A1. Ship and Submarine Tracks from Simulation 1

TABLE A1: COURSE, SPEED, CONTACT DATA FROM SIMULATION 1

TURN	1	2	3	4	5	6	7	8	9	10
CONTACT	NO	NO	NO	YES	YES	NO	YES	YES	YES	YES
OFFSET	0	0	0	30	-90	0	30	-90	60	180
SPEED	15	15	15	24	16	15	8	24	20	16
ATTACK	NO	NO	NO	NO	NO	NO	NO	NO	YES	NO
SHIP CSE	80	80	80	80	80	80	80	80	80	80
SHIP SPD	15	15	15	15	15	15	15	15	15	15

TURN	11	12	13	14	15	16	17	18	19	20
CONTACT	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
OFFSET	-90	30	60	60	30	30	-90	60	0	60
SPEED	8	24	8	4	20	24	16	8	12	8
ATTACK	YES	NO	NO	YES	NO	NO	NO	NO	YES	NO
SHIP CSE	80	80	80	80	80	80	80	80	80	80
SHIP SPD	15	5	5	5	5	5	5	5	5	5

TURN	21	22	23	24	25	26	27	28	29	30
CONTACT	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
OFFSET	180	30	0	0	30	60	30	45	0	30
SPEED	16	20	15	16	8	12	4	12	16	12
ATTACK	YES	NO	NO	NO	NO	YES	YES	YES	NO	YES
SHIP CSE	80	80	80	80	80	80	80	80	80	80
SHIP SPD	5	5	5	5	5	5	5	5	5	5

TURN	31	32	33	34	35	36	37	38	39	40
CONTACT	YES	YES	NO	YES	YES	NO	NO	YES	NO	YES
OFFSET	0	-90	0	45	180	0	0	45	0	45
SPEED	16	16	15	12	4	15	15	24	15	12
ATTACK	NO	NO	NO	YES	NO	NO	NO	NO	NO	YES
SHIP CSE	80	80	80	80	80	80	80	80	80	80
SHIP SPD	5	5	5	5	5	5	5	5	5	5

TURN	41	42	43
CONTACT	YES	YES	YES
OFFSET	0	45	60
SPEED	16	16	22
ATTACK	NO	YES	YES
SHIP CSE	80	80	80
SHIP SPD	5	5	5

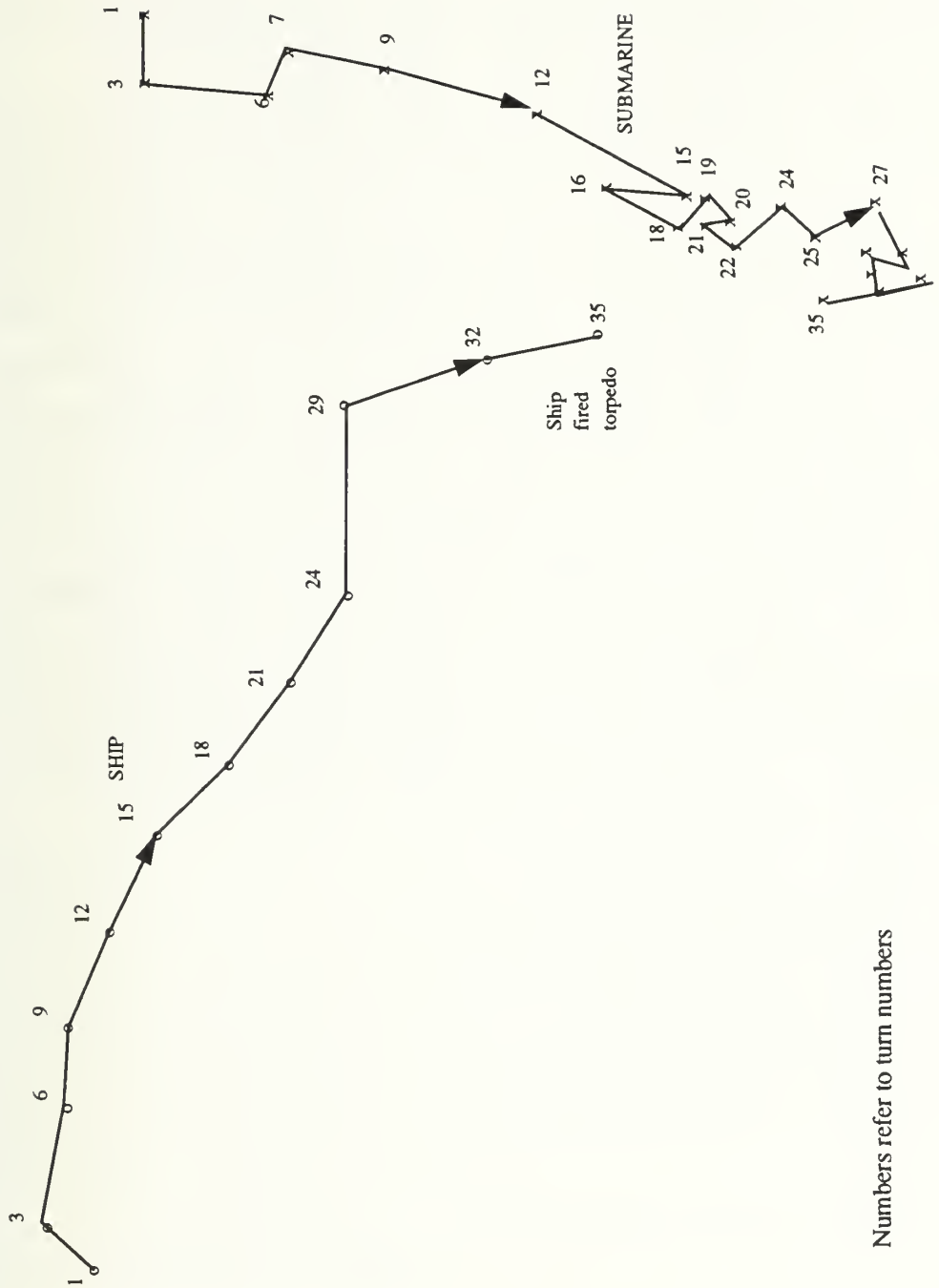

Figure A2. Ship and Submarine Tracks from Simulation 2

TABLE A2: COURSE, SPEED, CONTACT DATA FROM SIMULATION 2

TURN	1	2	3	4	5	6	7	8	9	10
CONTACT	NO	NO	NO	YES	YES	YES	YES	YES	YES	NO
OFFSET	0	0	0	-90	-90	-90	180	-90	-90	0
SPEED	15	15	15	16	16	16	20	24	24	15
ATTACK	NO	NO	NO	NO	NO	NO	YES	YES	NO	NO
SHIP CSE	080	050	050	094	094	094	094	094	094	110
SHIP SPD	15	15	15	15	15	15	15	15	15	15

TURN	11	12	13	14	15	16	17	18	19	20
CONTACT	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
OFFSET	-90	-90	-90	-90	-90	60	60	-90	-90	180
SPEED	24	24	24	24	24	24	4	8	24	16
ATTACK	NO	NO	NO	NO	NO	YES	NO	YES	NO	YES
SHIP CSE	110	110	110	125	125	125	125	125	125	125
SHIP SPD	15	15	15	15	15	15	15	15	15	15

TURN	21	22	23	24	25	26	27	28	29	30
CONTACT	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
OFFSET	-90	45	60	-90	180	180	-90	180	180	-90
SPEED	12	12	8	24	16	12	24	16	12	24
ATTACK	YES	YES	YES	NO	YES	NO	NO	YES	NO	NO
SHIP CSE	125	125	125	125	125	125	090	090	090	155
SHIP SPD	15	15	15	15	15	15	15	25	25	25

TURN	31	32	33	34	35
CONTACT	YES	YES	YES	YES	YES
OFFSET	60	-90	90	0	0
SPEED	20	24	22	24	24
ATTACK	NO	YES	YES	NO	NO
SHIP CSE	155	155	166	166	166
SHIP SPD	15	15	15	15	15

APPENDIX B

A. INSTRUCTIONS FOR PLAYING THE SIMULATOR

1. Enter the directory which contains the simulator, and enter the command "main_game." This will initiate the game. The user is then asked if he/she chooses to play or to exit. If the choice is made to play, the initialization procedures are performed.

2. After their completion, the user is shown the opening information (i.e. initial datum, ship's position, course and speed). The user is then prompted for any changes that may be desired. The same format used here is repeated at the beginning of each turn, as the user makes any inputs. Course, speed, and depth changes must be entered as float values. (The depth choices for the towed-array are listed. Maximum speed is 29.0 knots.) If improper format is used to enter the information, the user is requested to try again. If the decision is made to attack, the user is asked to give range and bearing information. (To simulate the use of aircraft, there are no restrictions here.) The horizontal difference between the torpedo and the submarine must be less than two miles for a chance at a hit. If this condition is met, a random number versus hit probability determines if the attack was a success.

3. The user will be given contact data on the two frequencies (300.0 Hertz and 1200.0 Hertz) being searched. If in contact, the bearing and frequency (after being adjusted for doppler) will be given. If the submarine is at periscope depth, the surface ship has a chance at detection via radar, visually, or ESM gear. If successful, the range and bearing (for radar or visual contact) or bearing (for ESM contact) information will be given to the user.

4. The game will continue until either ship or submarine is hit, or the user decides to exit the game. This option is given each turn.

5. After exiting, the user is given a screen printout of the course, speed, depth, position, et cetera information from each turn.

APPENDIX C

TABLE A3: PAYOFF VALUES

Submarine's Goal	Destroy the Surface Combatant				
Substate	Action	Action Worth			
		1	2	3	4
Attack	attack	1	3	7	9
	do not attack	0	0	0	0
Change Course (degrees)	45	10	12	14	15
	30	8	10	12	14
	60	7	8	10	10
	0	6	6	8	8
	90	5	5	5	5
	-90	4	4	4	4
	180	3	3	3	3
New Speed (knots)	8	10	10	10	10
	12	7	7	7	7
	16	4	4	4	4
	4	4	4	4	4
	20	2	2	2	2
	24	1	1	1	1
New Depth (ft)	150	8	8	8	8
	no change	9	9	9	9
	60	8	8	10	9
	300	6	6	6	6
	450	2	2	2	2
	600	1	1	7	1

APPENDIX D

he computer code is divided into packages, with the specifications and the bodies next
o each other. Beginning with the main procedure, the packages are listed in the order
hey are called by the main procedure. The data packages are listed last.

```
-Title:        main_game.a
-Subject:      This runs the simulator by calling the procedures from
-              the appropriate packages.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

ith ENV_DATA, DATA, INITIALIZE, DECIDE, DETECTION, GENALG, END_TURN, TEXT_IO;

se TEXT_IO;

rocedure MAIN_GAME is

 package INTEGER_INOUT is new INTEGER_IO (INTEGER);
 package FLOAT_INOUT is new FLOAT_IO (FLOAT);

 use INTEGER_INOUT, FLOAT_INOUT;

 NUM_STRINGS     :  INTEGER := INITIALIZE.NUM_STRINGS;  --The number of
                                                        --strings in the population
 PLAY            :  BOOLEAN;       --Holds user decision on continuing game
 SHIP_RECORD     :  DATA.SHIP_DATA; --Holds data on ship (i.e. course, speed)
 SUB_RECORD      :  DATA.SUB_DATA;  --Holds data on sub (i.e. course, speed)
 NEW_TURN        :  DATA.TURN_RECORD; --Holds historical data for post-game review
 PROB            :  DATA.PROB_RANGE;  --Holds range/probability data
 THIS_ENV        :  DATA.ENV_CHOICE;  --Used to hold seven values of current environ.
 NEW_ERROR       :  DETECTION.ERROR_RECORD;--Holds "errors" used in determining detec.
 WMOVE           :  INITIALIZE.MOVE_VALUE;--Holds point values for 504 tactic combos
 MOVE            :  INITIALIZE.MOVE_NUM;--Array of move numbers for each tactic combo
 WENV            :  INITIALIZE.ENV_VALUE;--Array of point values for each of 512 envir.
 ENV             :  INITIALIZE.ENV_NUM;--Associates numberr with each environ combo
 KGPOOL          :  INITIALIZE.BIT_STRINGS;--Initial population
 TIMER           :  DATA.TIME_RECORD;   --Holds time and turn info
 IELAPT          :  INTEGER := 1;     --elapsed time for game
 ITSTEP          :  INTEGER := 3;     --the length of each turn (in minutes)
 KGTURN          :  INTEGER := 1;     --the turn number
 F2,
 SUB_FREQ        :  ENV_DATA.FREQ_ARRAY := ENV_DATA.NEW_FREQS;--Freqs for detection
 DEC             :  GENALG.DECI_FIT := (others => (others => 0.0));
 SUB_HIT         :  BOOLEAN  := FALSE;
 FIVE_ENV        :  GENALG.OLD_ENV  :=  (320, 316, 316, 316, 316);--initial queue of
 SHIP_SUNK       :  BOOLEAN  := FALSE;                            --environments for
                                                                  --grading strings
egin

 --Sets up the game
 --Runs all the procedures in the INITIALIZE package to build the bit
 --strings, display data, determine space and environment values, etc.
 INITIALIZE.SET_UP (WMOVE, WENV, ENV, WJSS, SHIP_RECORD, SUB_RECORD,
                    TIMER,KGPOOL, PLAY, MOVE, NUM_STRINGS);

 --As long as the user wants to continue and the time has not run out.  Game will end
 --after 500 turns (1500 minutes)
 while (PLAY and (IELAPT < 1500)) loop
   PUT ("TURN ");
   PUT (KGTURN, WIDTH => 1);
   NEW_LINE;
```

```
    --Determines the prob of detection, if the ship or sub detects
    --the other, and outputs the results.
    DETECTION.RUN_DETECTION (IELAPT, ITSTEP, KGTURN,
                             SHIP_RECORD,SUB_RECORD,NEW_ERROR, NEW_TURN,
                             F2, TIMER, THIS_ENV, PLAY);
    --Continue if desired
    if PLAY then
       --The ship decides whether or not to change tactics, and if to attack,
      --followed by the sub's decision (using the genetic algorithm) on
      --its tactics.
      DECIDE.RUN_DECIDE (SUB_RECORD, SHIP_RECORD, NEW_TURN,PROB,
                         KGTURN, KGPOOL,MOVE, THIS_ENV, SUB_HIT, ENV,
                         DEC, WMOVE, WENV,NUM_STRINGS, FIVE_ENV,
                         SHIP_SUNK);
      --Updates the clock, and the ship and sub positions
      END_TURN.RECORD_UPDATE (SHIP_RECORD, SUB_RECORD, NEW_TURN,
                              KGTURN,WENV, WMOVE);
      END_TURN.TIME_AND_TURN (IELAPT, TIMER, KGTURN);
      --If the target has been hit, the game will cease

      --If the ship-to-sub distance exceeds 40 nm, the game ends
      if SUB_RECORD.SUB_RANGE > 40.0 then
        exit;
      end if;

      if SUB_HIT then
        PUT_LINE ("The submarine has been sunk.  Thank you for playing.");
        exit;
      end if;

     if SHIP_SUNK then
       PUT_LINE ("The ship has been sunk.  Thank you for playing.");
       exit;
     end if;

    --The user does not want to continue
    else
      PUT_LINE ("Thanks for playing.");
      NEW_LINE;
      exit;
    end if;
  end loop;
  --Provides the user with a debrief if desired
  END_TURN.WASH_UP(NEW_TURN, KGTURN);

end MAIN_GAME;
```

```
--Title              :  initialize_s.a
--Subject            :  Contains data and procedures to set up ASW simulator
--------------------------------------------------------------------------
--------------------------------------------------------------------------
with DATA;

package INITIALIZE is

  NUM_STRINGS  :  INTEGER  := 8;--The population size
  type ARRAY_4 is array (1..4) of float;
  type ARRAY_2 is array (1..2) of float;
  type ARRAY_6 is array (1..6) of float;
  type ARRAY_7 is array (1..7) of float;
  type MOVE_VALUE is array (1..504) of float;
  type MOVE_NUM is array (1..2, 1..7, 1..6, 1..6) of integer;
  type ENV_VALUE is array (1..512) of FLOAT;
  type ENV_NUM is array (1..4, 1..4, 1..2, 1..2, 1..2, 1..2, 1..2) of integer;
  type BIT_STRINGS is array (1..NUM_STRINGS, 1..512) of integer;

  --Updates ship/sub position after a turn
  procedure XYSTEP (CSE,
                    SPD       : in FLOAT;
                    XSTEP,
                    YSTEP     : out FLOAT);

  --Calculates the space values for the move options of the submarine
  procedure MOVE_STATE_SPACE (TEMP_WMOVE : out MOVE_VALUE;
                              MOVE       : in out MOVE_NUM);


  --Calculates the space values for the various environments (in reference
  --to the surface ship) in which the sub may find itself.
  procedure ENV_STATE_SPACE (TEMP_WENV : out ENV_VALUE;
                             ENV       : out ENV_NUM);

  --Fills the initial bit strings for use in the program
  procedure INIT_BIT_STRINGS (KGPOOL      : out BIT_STRINGS;
                              NUM_STRINGS : in INTEGER);

  --Asks the user if he wants to play or not B1
  procedure PLAY_OR_NOT (PLAY: out BOOLEAN);

  --Displays the initial values and asks the user if he wants to make changes
  procedure DISPLAY (SHIP_RECORD : in out DATA.SHIP_DATA;
                     SUB_RECORD  : in out DATA.SUB_DATA;
                     TIMER       : in out DATA.TIME_RECORD);

  --Runs the above procedures to initialize the data, etc. for game play
  procedure SET_UP (WMOVE          : in out MOVE_VALUE;
                    WENV ,
                    ENV            : in out ENV_NUM;
                    SHIP_RECORD    : out DATA.SHIP_DATA;
                    SUB_RECORD     : out DATA.SUB_DATA;
                    TIMER          : out DATA.TIME_RECORD;
                    KGPOOL         : in out BIT_STRINGS;
                    PLAY           : in out BOOLEAN;
                    MOVE           : in out MOVE_NUM;
                    NUM_STRINGS    : in INTEGER);

end INITIALIZE;
```

```
--Title              :   initialize_b.a
--Subject            :   Contains data and procedures to set up ASW simulator
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
with TEXT_IO, MATH_LIB, CALENDAR, U_Rand;
use TEXT_IO, MATH_LIB, CALENDAR;

package body INITIALIZE is

  --Used in U_RAND generator
  U : NATURAL;
  K : constant := 5**5;
  M : constant := 2**13;

  package INTEGER_INOUT is new INTEGER_IO (INTEGER);
  package FLOAT_INOUT is new FLOAT_IO (FLOAT);

  use INTEGER_INOUT, FLOAT_INOUT;

  --Determines logorithms in base 10 (this is not in MATH_LIB)
  function MY_LOG (NUM : FLOAT) return FLOAT is

    NAT_LOG_10     : constant :=  2.30258_5;
    LOG_10         : FLOAT;

  begin
    LOG_10 := (MATH_LIB.LN (NUM))/NAT_LOG_10;
    return LOG_10;
  end MY_LOG:

  --Determines the position of the ship/sub at the end of a turn
  procedure XYSTEP (CSE,
                    SPD   : in FLOAT;
                    XSTEP,
                    YSTEP : out FLOAT) is

    MY_PI : FLOAT := 3.14159;
    PI_12    : FLOAT := MY_PI/2.0;
    PI_32    : FLOAT := 3.0*MY_PI/2.0;
    TSTEP : FLOAT := 3.0;          --3 minute turn period
    RDCSE : FLOAT;                 --Course in radians
    DIST  : FLOAT;                 --Distance
    RANG  : FLOAT;                 --Normalized radian course

  begin
    --Convert course from degrees to radians
    RDCSE := CSE * MY_PI/180.0;
    --Determine distance traveled during the timestep
    DIST := SPD * (TSTEP/60.0);

    --Convert the cse/spd to X & Y changes, based on what quadrant the course falls in
    if RDCSE <= PI_12 then
      XSTEP := MATH_LIB.COS (RDCSE) * DIST;
      YSTEP := MATH_LIB.SIN (RDCSE) * DIST;

    elsif RDCSE <= MY_PI then
       RANG := RDCSE - PI_12;
       XSTEP := MATH_LIB.COS (RANG) * DIST;
       YSTEP := -1.0 * (MATH_LIB.SIN (RANG) * DIST);

    elsif RDCSE <= PI_32 then
      RANG := PI_32 - RDCSE;
      XSTEP := -1.0 * (MATH_LIB.COS (RANG) * DIST);
      YSTEP := -1.0 * (MATH_LIB.SIN (RANG) * DIST);
```

```
  else
    RANG := RDCSE - PI_32;
    XSTEP := -1.0 * (MATH_LIB.COS(RANG) * DIST);
    YSTEP := MATH_LIB.SIN (RANG) * DIST;
  end if;

end XYSTEP;

--Calculates the space values for the move options of the submarine
procedure MOVE_STATE_SPACE (TEMP_WMOVE : out MOVE_VALUE;       --1x504
                            MOVE       : in out MOVE_NUM) is

  IAU                       : INTEGER := 1;       --Counter for looping through matrix
  AU,
  BU,
  CU,
  DU                        : FLOAT;                  --Hold values for matrix entry
  DUPSV                     : FLOAT := 0.0;          --Holds sum of space values
  TUPSV                     : FLOAT := 0.0;         --Holds value for particular move
  NEW_DATA                  : DATA.ENV_MOVE_VALUES;      --Values for sub tactics

begin
  --Total maximum space value (based on optimum move choice by sub)
  for I in 1..4 loop
    DUPSV := DUPSV + NEW_DATA.UPSV (I);
  end loop;

  --UPSV and UMW contain the values for the various tactics the submarine
  --can execute.  AU, BU, CU, DU are temporary variables used below to
  --enter the values into the value matrix.
  --Space value / move weight
  AU := NEW_DATA.UPSV (1)/NEW_DATA.UMW (1);
  BU := NEW_DATA.UPSV (2)/NEW_DATA.UMW (2);
  CU := NEW_DATA.UPSV (3)/NEW_DATA.UMW (3);
  DU := NEW_DATA.UPSV (4)/NEW_DATA.UMW (4);

  --IB, IC, ID, IE are counters used to reach every possible move
  --combination for the sub.  UC1(attack), UC2(course), UC3(speed),
  --UC4(depth) are the arrays containing the options for each area.
  --WMOVE contains the computed value for a given combination of tactics.
  --IAU counts through the combinations (504 total).
  for IB in 1..2 loop
    for IC in 1..7 loop
      for ID in 1..6 loop
        for IE in 1..6 loop
          --The total space value for this particular combination of moves
          TUPSV := (AU* NEW_DATA.UC1(IB)) + (BU* NEW_DATA.UC2(IC))
                +  (CU* NEW_DATA.UC3(ID)) + (DU* NEW_DATA.UC4(IE));
           --The percentage of this move's values of the maximum possible
           TEMP_WMOVE(IAU) := TUPSV/DUPSV;
           --Move number associated with this set of moves
           MOVE (IB, IC, ID, IE) := IAU;
           --Increment the counter
           IAU := IAU + 1;
        end loop;
      end loop;
    end loop;
  end loop;

end MOVE_STATE_SPACE;

--Calculates the space values for the various environments (in reference
--to the surface ship) in which the sub may find itself.
procedure ENV_STATE_SPACE (TEMP_WENV : out ENV_VALUE;    --1x512
```

```
                                                    ENV                 : out ENV_NUM) i

  AV, BV, CV, DV, EV, FV, GV : FLOAT;
  DVPSV                                          : FLOAT;
  TVPSV                                           : FLOAT;
  IAV                                                : INTEGER;
  NEW_DATA                                 : DATA.ENV_MOVE_VALUES;

begin
  --The total points available
  for I in 1..7 loop
    DVPSV := DVPSV +  NEW_DATA.VPSV(I);
  end loop;

  --AV-GV are temp variables used below.  They contain the value for a
  --given environment (combination of data the sub has on the ship)
  --Space value/move weight
  AV :=  NEW_DATA.VPSV (1)/ NEW_DATA.VMW (1);
  BV :=  NEW_DATA.VPSV (2)/ NEW_DATA.VMW (2);
  CV :=  NEW_DATA.VPSV (3)/ NEW_DATA.VMW (3);
  DV :=  NEW_DATA.VPSV (4)/ NEW_DATA.VMW (4);
  EV :=  NEW_DATA.VPSV (5)/ NEW_DATA.VMW (5);
  FV :=  NEW_DATA.VPSV (6)/ NEW_DATA.VMW (6);
  GV :=  NEW_DATA.VPSV (7)/ NEW_DATA.VMW (7);
  --IAV counts through the 512 possible environments
  IAV := 1;

  --All 512 environments are given their point value, based on the combination
  --of contact freq(IB), contact range (IC), doppler (ID), bearing drift (IE),
  --bearing trend (IFF), bearing rate (IG), contact strength(IH)
  for IB in 1..4 loop
    for IC in 1..4 loop
      for ID in 1..2 loop
        for IE in 1..2 loop
          for IFF in 1..2 loop
            for IG in 1..2 loop
              for IH in 1..2 loop
                --Value of this set of environmental inputs
                TVPSV := (AV* NEW_DATA.VC1(IB)) + (BV* NEW_DATA.VC2(IC)) +
                         (CV* NEW_DATA.VC3(ID)) + (DV* NEW_DATA.VC4(IE)) +
                         (EV* NEW_DATA.VC5(IFF)) + (FV* NEW_DATA.VC6(IG))+
                         (GV* NEW_DATA.VC7(IH));
                --Total points/this combination's points
                TEMP_WENV(IAV) := (TVPSV/DVPSV);
                --Assigns a number to this particular environment
                ENV (IB, IC, ID, IE, IFF, IG, IH) := IAV;
                IAV := IAV + 1;
              end loop;      --IH
            end loop;        --IG
          end loop;          --IFF
        end loop;            --IE
      end loop;              --ID
    end loop;                --IC
  end loop;                  --IB

end ENV_STATE_SPACE;

--Fills the initial bit strings for use in the program
procedure INIT_BIT_STRINGS (KGPOOL : out BIT_STRINGS;
                            NUM_STRINGS : in INTEGER) is   --8x512

  XM,
  XXX    : FLOAT;
  LGPOOL : BIT_STRINGS;                        --used for debugging
```

```
begin
  --Gives a random value to each allele in the initial population
  for IX in 1..NUM_STRINGS loop
    --PUT_LINE ("INIT.BIT_STRINGS 2 ");
    --each string has 512 alleles
    for IY in 1..512 loop
      --A value from 0 to 504 will be assigned to each allele at random
      --and entered into the KGPOOL matrix which holds the bit strings
      XXX := U_Rand.Next ;
      --Picks the move choice at random
      XM := (XXX * 503.0) + 1.0;
      --Assigns the value
      KGPOOL (IX, IY) := INTEGER (XM);
      LGPOOL (IX, IY) := INTEGER (XM);
    end loop;       --IY
  end loop;         --IX

end INIT_BIT_STRINGS;

--Asks the user if he wants to play or not
procedure PLAY_OR_NOT (PLAY: out BOOLEAN) is

  CHOICE : CHARACTER;
  WRONG_ENTRY : BOOLEAN := TRUE;

begin
  --PUT_LINE ("INITIALIZE.PLAY 4 ");
  while WRONG_ENTRY loop
    WRONG_ENTRY := FALSE;
    PUT ("Do you want to execute the game or exit the system?");
    NEW_LINE;
    PUT ("Please enter '1' to execute or '2' to exit.");
    NEW_LINE;
    GET (CHOICE);
    SKIP_LINE;
    CHOICE := '1';
    if CHOICE = '1' then
      PLAY := TRUE;
    elsif CHOICE = '2' then
      PLAY := FALSE;
    else
      PUT_LINE ("Incorrect entry.  Please try again.");
      WRONG_ENTRY := TRUE;
    end if;    --CHOICE
  end loop;      --WRONG_ENTRY
end PLAY_OR_NOT;

--Displays the initial values and asks the user if he wants to make changes
procedure DISPLAY (SHIP_RECORD : in out DATA.SHIP_DATA;
                   SUB_RECORD  : in out DATA.SUB_DATA;
                   TIMER       : in out DATA.TIME_RECORD) is

  START_DATA : DATA.SET_UP_DATA;
  NEW_CHANGE : BOOLEAN := TRUE;
  CHANGE     : CHARACTER;
  CHOICE     : CHARACTER;
  ZI         : FLOAT;
  XSTEP      : FLOAT;
  YSTEP      : FLOAT;
  WRONG_ENTRY : BOOLEAN := FALSE;
  GET_CHANGE  : BOOLEAN := TRUE;

begin
```

```
PUT_LINE ("The scenario contains the following inputs: ");
PUT_LINE("GAMETIME: ");
PUT ("DAY:       ");
PUT (TIMER.GDAY); NEW_LINE;
PUT ("HOUR:     ");
PUT (TIMER.GHR); NEW_LINE;
PUT ("MINUTES: ");
PUT (TIMER.GMIN); NEW_LINE;
PUT ("MONTH : ");
PUT (TIMER.MONTH); NEW_LINE;
PUT_LINE ("The grid is a 500NM by 500NM square.");
PUT_LINE ("Initial Sub Datum: ");
PUT ("X(NM) = ");
PUT (SUB_RECORD.XU, FORE => 5, AFT => 2, EXP => 0);
PUT ("  Y(NM) = ");
PUT (SUB_RECORD.YU, FORE => 5, AFT => 2, EXP => 0);
NEW_LINE;
PUT_LINE("Initial Ship Position: ");
PUT ("X(NM) = ");
PUT (SHIP_RECORD.XV, FORE => 5, AFT => 2, EXP => 0);
PUT ("  Y(NM) = ");
PUT (SHIP_RECORD.YV, FORE => 5, AFT => 2, EXP => 0);
NEW_LINE;
PUT ("Ocean area: ");
PUT (START_DATA.OCEAN);
NEW_LINE;
PUT ("Sub Class: ");
PUT (START_DATA.SUBCL);
NEW_LINE;
PUT ("Ship class and sensor: ");
PUT (START_DATA.SHIPCL);
NEW_LINE;
PUT ("ROE: ");
PUT (START_DATA.ROE);
NEW_LINE;
PUT_LINE ("Weapons status is free.");
PUT_LINE ("Current ship information: ");
PUT ("Course: ");
PUT (SHIP_RECORD.VCSE, FORE => 5, AFT => 1, EXP => 0);
NEW_LINE;
PUT ("Speed:  ");
PUT (SHIP_RECORD.VSPD, FORE => 5, AFT => 1, EXP => 0);
NEW_LINE;
PUT ("Array depth: ");
PUT (SHIP_RECORD.ZV, FORE => 5, AFT => 1, EXP => 0);
NEW_LINE;
PUT ("Not in contact.");
NEW_LINE;
PUT ("Engagement Status : ");
PUT (START_DATA.FTORPAU);
NEW_LINE;

while WRONG_ENTRY loop
  WRONG_ENTRY := FALSE;

 while GET_CHANGE loop
   GET_CHANGE := FALSE;
   PUT ("Any changes? 1-Yes, 2-No");
   NEW_LINE;
   GET (CHANGE);
   SKIP_LINE;

   --If changes are not desired
   if CHANGE = '2' then
```

```
        NEW_CHANGE := FALSE;
    elsif CHANGE = '1' then
        NEW_CHANGE := TRUE;
    else
       PUT_LINE ("Improper format.  Try again");
       GET_CHANGE := TRUE;
    end if;     --CHANGE
  end loop;     --GET_CHANGE

  while NEW_CHANGE loop
    if CHANGE = '1' then
    begin

      PUT_LINE ("Select 1-Course, 2-Speed, 3-Array depth");
      GET (CHOICE);

      if CHOICE = '1' then
        PUT_LINE ("Enter new course: XXX.X degrees true");
        GET (SHIP_RECORD.VCSE);
      elsif CHOICE = '2' then
        PUT_LINE ("Enter new speed: XX.X knots (Max of 29.0)");
        GET (SHIP_RECORD.VSPD);
      elsif CHOICE = '3' then
        PUT_LINE ("Enter array depth:  90.0, 150.0, 300.0, 450.0 feet");
        GET (SHIP_RECORD.ZV);
      else
        PUT_LINE ("That wasn't a choice.");
      end if;      --CHOICE

      exception
        when DATA_ERROR =>
          PUT_LINE ("IMPROPER ENTRY.  TRY AGAIN.");
          WRONG_ENTRY := TRUE;
        end;

       PUT_LINE ("Any more changes? If no, enter 1.  If yes, enter 2.");
       GET (CHOICE);

       if CHOICE = '1' then
           NEW_CHANGE := FALSE;
       end if;
     end if;      --CHANGE = '1'
   end loop;      --NEW_CHANGE

  end loop;    --WRONG_ENTRY

  --Determines ship and movement for this time period
  XYSTEP (SHIP_RECORD.VCSE, SHIP_RECORD.VSPD, XSTEP, YSTEP);
  SHIP_RECORD.XV := SHIP_RECORD.XV + XSTEP;
  SHIP_RECORD.YV := SHIP_RECORD.YV + YSTEP;

  XYSTEP (SUB_RECORD.UCSE, SUB_RECORD.USPD, XSTEP, YSTEP);
  SUB_RECORD.XU := SUB_RECORD.XU + XSTEP;
  SUB_RECORD.YU := SUB_RECORD.YU + YSTEP;

end DISPLAY;



--Runs the above procedures to initialize the data, etc. for game play
procedure SET_UP (WMOVE         : in out MOVE_VALUE;      --1x504
                  WENV          : in out ENV_VALUE;       --1x512
                  ENV           : in out ENV_NUM;
                  SHIP_RECORD   : out DATA.SHIP_DATA;
```

```
                    SUB_RECORD     : out DATA.SUB_DATA;
                    TIMER          : out DATA.TIME_RECORD;
                    KGPOOL         : in out BIT_STRINGS;     --NUM_STRINGSx512
                    PLAY           : in out BOOLEAN;
                    MOVE           : in out MOVE_NUM;
                    NUM_STRINGS    : in INTEGER) is

    TEMP_SHIP_RECORD : DATA.SHIP_DATA;
    TEMP_SUB_RECORD  : DATA.SUB_DATA;
    TEMP_TIMER       : DATA.TIME_RECORD;

  begin
    --Determines if the user wants to play
    PLAY_OR_NOT (PLAY);

    if PLAY then

      --Calculate values for the move options of the sub
      MOVE_STATE_SPACE (WMOVE, MOVE);
      --Calculate values for the environments the sub is in
      ENV_STATE_SPACE (WENV, ENV);
      --Calculates the joint space values
      --Creates the initial bit strings
      INIT_BIT_STRINGS ( KGPOOL, NUM_STRINGS);
      --Output initial data
      DISPLAY (TEMP_SHIP_RECORD, TEMP_SUB_RECORD, TEMP_TIMER);
      SHIP_RECORD := TEMP_SHIP_RECORD;
      SUB_RECORD  := TEMP_SUB_RECORD;
      TIMER       := TEMP_TIMER;

    --Play is not desired
    else
      PUT_LINE ("Thank you for turning me on.");
    end if;     --if PLAY
  end SET_UP;

end INITIALIZE;
```

```
-Title:      detection_s.a
-Subject:    Contains the procedures which determine if detection has
-            been achieved
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

ith DATA, ENV_DATA;

ackage DETECTION is

 --Contains the adjustments to the times for determining detection
 type ERROR_RECORD is
   record
     LAMCMT : FLOAT := 0.0;                  --lambda mean time
     LAMCM  : FLOAT := 1.0;                  --lambda mean
     LAMCME : FLOAT := 0.0;                  --lambda mean corrected
     SIGCM  : FLOAT := 5.0;                  --sigma mean
     LAMVT  : FLOAT := 0.0;                  --ship lambda time
     LAMV   : FLOAT := 1.0;                  --lambda ship
     LAMVE  : FLOAT := 0.0;                  --lambda ship corrected
     SIGV   : FLOAT := 3.0;                  --ship sigma
     LAMUT  : FLOAT := 0.0;                  --lambda sub time
     LAMU   : FLOAT := 1.0;                  --lambda sub
     LAMUE  : FLOAT := 0.0;                  --lambda sub corrected
     LAMSUM : FLOAT := 0.0;                  --sum of lambdas
     SIGU   : FLOAT := 3.0;                  --sub sigma
   end record;

 --Adds two sound levels together
 procedure PWRSUM (BN1,
                   BN2  : in FLOAT;
                   BNNL : out FLOAT);


 --Calculates the bearing between two positions
 procedure BRGCLC (X1,
                   Y1,
                   X2,
                   Y2     : in FLOAT;
                   CONTBR : out FLOAT);


 --The error terms are added to the sonar equation to create
 --randomness.  The time periods are also determined.  Each
 --time period is added to the cumulative time for that error
 --term.  If the elapsed time is less than the error time, the
 --error term is not changed.  If it is greater, a new error term is
 --added.
 procedure DETECT_VARIABLES (IELAPT      : in INTEGER;
                             SHIP_RECORD : in out DATA.SHIP_DATA;
                             SUB_RECORD  : in out DATA.SUB_DATA;
                             NEW_ERROR   : in out ERROR_RECORD;
                             NEW_TURN    : in out DATA.TURN_RECORD;
                             KGTURN      : in INTEGER);


 --Using the sonar equation, this determines whether or not the sub
 --will gain contact on the surface ship
 procedure SUB_DETECT (SHIP_RECORD   : in out DATA.SHIP_DATA;
                       SUB_RECORD    : in out DATA.SUB_DATA;
                       NEW_ERROR     : in out ERROR_RECORD;
                       NEW_TURN      : in out DATA.TURN_RECORD;
                       ITSTEP,
                       KGTURN        : in INTEGER);
```

```
   --Determines the current environment valuated state space
   procedure CURRENT_ENV (SUB_RECORD  : in out DATA.SUB_DATA;
                          NEW_TURN    : in out DATA.TURN_RECORD;
                          KGTURN      : in INTEGER;
                          THIS_ENV    : in out DATA.ENV_CHOICE);


   --Determines if the surface ship can detect the sub
   procedure SHIP_DETECT (F2              : out ENV_DATA.FREQ_ARRAY;
                          SHIP_RECORD     : in out DATA.SHIP_DATA;
                          SUB_RECORD      : in out DATA.SUB_DATA;
                          NEW_ERROR       : in out ERROR_RECORD);



   --Outputs the detection results
   procedure DETECT_RESULTS (SHIP_RECORD  : in out DATA.SHIP_DATA;
                             SUB_RECORD   : in out DATA.SUB_DATA;
                             TIMER        : in DATA.TIME_RECORD;
                             F2           : in ENV_DATA.FREQ_ARRAY;
                             PLAY         : in out BOOLEAN);


   --Runs the above procedures
   procedure RUN_DETECTION (IELAPT,
                            ITSTEP,
                            KGTURN      : in INTEGER;
                            SHIP_RECORD : in out DATA.SHIP_DATA;
                            SUB_RECORD  : in out DATA.SUB_DATA;
                            NEW_ERROR   : in out ERROR_RECORD;
                            NEW_TURN    : in out DATA.TURN_RECORD;
                            F2          : in out ENV_DATA.FREQ_ARRAY;
                            TIMER       : in DATA.TIME_RECORD;
                            THIS_ENV    : in out DATA.ENV_CHOICE;
                            PLAY        : in out BOOLEAN);

end DETECTION;
```

```
-Title:      detection_b.a
-Subject:    Contains the procedures which determine if detection has
-            been achieved
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

ith TEXT_IO, CALENDAR, INITIALIZE,U_Rand, MATH_LIB;
se TEXT_IO, CALENDAR, INITIALIZE, MATH_LIB;

ackage body DETECTION is

 package FLOAT_INOUT is new FLOAT_IO (FLOAT);
 package INTEGER_INOUT is new INTEGER_IO (INTEGER);

 use FLOAT_INOUT, INTEGER_INOUT;

 --Determines base 10 log (not available in MATH_LIB)
 function MY_LOG (NUM : FLOAT) return FLOAT is

   NAT_LOG_10     : constant :=  2.30258_5;
   LOG_10         : FLOAT;

 begin
   LOG_10 := (MATH_LIB.LN (NUM))/NAT_LOG_10;
   return LOG_10;
 end MY_LOG;


 --Adds two sound levels together
 procedure PWRSUM (BN1,
                   BN2  : in FLOAT;
                   BNNL : out FLOAT) is

   type ARRAY_10 is array (0..10) of FLOAT;

   --Based on the difference between two levels, one of these values will
   --be added to the greater level for the result
   PSUM : ARRAY_10 := (3.0,2.5,2.1,1.7,1.4,1.2,1.0,0.8,0.6,0.5,0.4);
   PW   : FLOAT;     --Used to determine array position
   IPW  : INTEGER;   --The integer conversion of PW

 begin
   --The difference in the two sound levels
   PW := ABS(BN1-BN2);

   --The maximum array position for levels with a difference greater than 10 db
   if PW > 10.0 then
     PW := 10.0;
   end if;

   IPW := INTEGER (PW);

   --The value from PSUM is added to the larger of the two inputs
   if BN1 <= BN2 then
     BNNL := BN2 + PSUM(IPW);
   else
     BNNL := BN1 + PSUM(IPW);
   end if;
 end PWRSUM;

 --Returns absolute value (not in MATH_LIB)
 function MY_ABS (X : FLOAT) return FLOAT is

   Y : FLOAT;
```

```
begin
  if (X < 0.0) then
    Y := -X;
  else
    Y := X;
  end if;
  return Y;
end MY_ABS;


--Calculates the bearing between two positions
procedure BRGCLC (X1,
                  Y1,
                  X2,
                  Y2      : in FLOAT;
                  CONTBR : out FLOAT) is

  MY_PI : FLOAT := 3.14159;
  PI_12 : FLOAT := MY_PI / 2.0;
  RD    : FLOAT := 57.2958;      --Degrees per radian
  RANG  : FLOAT;                 --Angle in radians
  DELX,
  DELY  : FLOAT;                 --The difference between positions
  Q,
  R,
  S     : FLOAT;

begin
  --The difference in X and Y between the positions
  DELX := X1 - X2;
  DELY := Y1 - Y2;
  R := MY_ABS (DELX);
  S := MY_ABS (DELY);

  --The ratio of Y change to X change
  Q := S/R;

  --For minimal Y changes, consider them to be 0
  if Q < 0.01 then
    DELY := 0.0;
  end if;

  --The cardinal bearings (either X or Y does not change)
  if (DELX = 0.0) OR (DELY = 0.0) then
    if (DELX = 0.0) AND (DELY < 0.0) then
      CONTBR := 0.0;
    elsif (DELX = 0.0) AND (DELY > 0.0) then
      CONTBR := 180.0;
    elsif (DELX < 0.0) AND (DELY = 0.0) then
      CONTBR := 90.0;
    elsif (DELX > 0.0) AND (DELY = 0.0) then
      CONTBR := 270.0;
    else
      CONTBR := 0.0;
    end if;
  else
    --Convert angle to Radians
    RANG := MATH_LIB.ATAN (Q);

    --The four quadrants are covered
    if (DELX < 0.0) AND (DELY < 0.0) then
      CONTBR := (PI_12 - RANG) * RD;
    elsif (DELX < 0.0) AND (DELY > 0.0) then
```

```
      CONTBR := (PI_12 + RANG) * RD;
    elsif (DELX > 0.0) AND (DELY > 0.0) then
      CONTBR := ((3.0 * PI_12) - RANG) * RD;
    else
      CONTBR := ((3.0 * PI_12) + RANG) * RD;
    end if;
  end if;

end BRGCLC;


--The error terms are added to the sonar equation to create
--randomness.  There time periods are also determined.  Each
--time period is added to the cumulative time for that error
--term.  If the elapsed time is less than the error time, the
--error term is not changed.  If it is greater, a new error term is
--added.
procedure DETECT_VARIABLES (IELAPT      : in INTEGER;
                            SHIP_RECORD : in out DATA.SHIP_DATA;
                            SUB_RECORD  : in out DATA.SUB_DATA;
                            NEW_ERROR   : in out ERROR_RECORD;
                            NEW_TURN    : in out DATA.TURN_RECORD;
                            KGTURN      : in INTEGER) is


  TIME   : float;   --holds time values while errors determined
  X,
  Y,
  Z,
  U1,   --random number
  U2,   --random number
  XXX,  --random number
  DB,
  B     : float;
  ELAPT : FLOAT;

begin

  --Elapsed time
  ELAPT := FLOAT(IELAPT);
  if ELAPT >= NEW_ERROR.LAMCMT then   --LAMCMT is the lambda mean time
    XXX := U_RAND.NEXT;
    TIME := (-1.0 * MY_LOG(XXX)) * NEW_ERROR.LAMCM;  --LAMCM is the lambda mean
    NEW_ERROR.LAMCMT := NEW_ERROR.LAMCMT + TIME;
    U1 := U_RAND.NEXT;
    U2 := U_RAND.NEXT;
    Z := MATH_LIB.SQRT(-2.0 * MY_LOG(U1)) * MATH_LIB.COS(6.2832 * U2);
    NEW_ERROR.LAMCME := NEW_ERROR.SIGCM * Z;
  end if;

  if ELAPT >= NEW_ERROR.LAMVT then     --LAMVT is the ship time lambda
    XXX := U_RAND.NEXT;
    TIME := (-1.0 * MY_LOG(XXX)) * NEW_ERROR.LAMV; --LAMV is the lambda ship
    NEW_ERROR.LAMVT := NEW_ERROR.LAMVT + TIME;
    U1 := U_RAND.NEXT;
    U2 := U_RAND.NEXT;
    Z := MATH_LIB.SQRT(-2.0 * MY_LOG(U1)) * MATH_LIB.COS(6.2832 * U2);
    NEW_ERROR.LAMVE := NEW_ERROR.SIGV * Z; --LAMVE is the lambda ship corrected
  end if;

  if ELAPT >= NEW_ERROR.LAMUT then      --LAMUT is the lambda sub time
    XXX := U_RAND.NEXT;
    TIME := (-1.0 * MY_LOG(XXX)) * NEW_ERROR.LAMU;  --LAMU is the lambda sub
    NEW_ERROR.LAMUT := NEW_ERROR.LAMUT + TIME;  --LAMUT is the sub lambda time
```

```
  U1 := U_RAND.NEXT;
  U2 := U_RAND.NEXT;
  Z := MATH_LIB.SQRT(-2.0 * MY_LOG(U1)) * MATH_LIB.COS(6.2832 * U2);
  NEW_ERROR.LAMUE := NEW_ERROR.SIGU * Z;   --lambda sub corrected
end if;

--Calculate the distance between the ship and the sub
X := SHIP_RECORD.XV - SUB_RECORD.XU;
Y := SHIP_RECORD.YV - SUB_RECORD.YU;
SUB_RECORD.SUB_RANGE   := MATH_LIB.SQRT ((X**2) + (Y**2));

--Update the history record
NEW_TURN.TR (KGTURN) := SUB_RECORD.SUB_RANGE;

--Determine submarine depth
if SUB_RECORD.ZU = 60.0 then
  SUB_RECORD.KD := 1;
elsif SUB_RECORD.ZU = 150.0 then
  SUB_RECORD.KD := 2;
elsif SUB_RECORD.ZU = 300.0 then
  SUB_RECORD.KD := 3;
elsif SUB_RECORD.ZU = 450.0 then
  SUB_RECORD.KD := 4;
elsif SUB_RECORD.ZU = 600.0 then
  SUB_RECORD.KD := 5;
else
  PUT_LINE ("Error in sub depth.");
end if;

--Determine surface receiver depth
if SHIP_RECORD.ZV = 90.0 then
  SHIP_RECORD.KH := 1;
elsif SHIP_RECORD.ZV = 150.0 then
    SHIP_RECORD.KH := 2;
elsif SHIP_RECORD.ZV = 300.0 then
    SHIP_RECORD.KH := 3;
elsif  SHIP_RECORD.ZV = 450.0 then
    SHIP_RECORD.KH := 4;
else
    PUT_LINE ("Error in receiver depth.");
end if;

--Calculate Theta, the difference between ship's course and the
--line of sound (of the contact)
if SHIP_RECORD.CTCBRG <= 180.0 then
  B := SHIP_RECORD.CTCBRG + 180.0;
else
  B := SHIP_RECORD.CTCBRG - 180.0;
end if;

--The difference in degrees
DB := B - SHIP_RECORD.VCSE;
--For "negative" figures
if SHIP_RECORD.VCSE > B then
  DB := 360.0 + DB;
end if;

--The difference in radians
SHIP_RECORD.TH := (DB/360.0) * 6.28318;   --Theta
NEW_TURN.TTH (KGTURN) := DB;

--Calculate PHI, the difference between sub's course and the
--line of sound
--The difference in degrees
```

```
  DB := SHIP_RECORD.CTCBRG - SUB_RECORD.UCSE;
  --For "negative" figures
  if (SUB_RECORD.UCSE > SHIP_RECORD.CTCBRG) then
    DB := 360.0 + DB;
  end if;
  --The difference in radians
  SUB_RECORD.PHI := (DB/360.0) * 6.28318;
  NEW_TURN.TPHI(KGTURN) := SUB_RECORD.PHI;

  --Calculate doppler range rate
  SUB_RECORD.DPLR :=(SHIP_RECORD.VSPD * MATH_LIB.COS(SHIP_RECORD.TH)) +
                    (SUB_RECORD.USPD * MATH_LIB.COS(SUB_RECORD.PHI));
  NEW_TURN.TDPLR(KGTURN) := SUB_RECORD.DPLR;

  if SUB_RECORD.DPLR < 0.0 then
    SUB_RECORD.DPLRF := 'D';      --Down Doppler
  elsif SUB_RECORD.DPLR > 0.0 then
    SUB_RECORD.DPLRF := 'U';      --Up doppler
  else
    SUB_RECORD.DPLRF := 'Z';      --No doppler
  end if;

end DETECT_VARIABLES;

--Using the sonar equation, this determines whether or not the sub will gain
--contact on the surface ship
procedure SUB_DETECT (SHIP_RECORD : in out DATA.SHIP_DATA;
                      SUB_RECORD  : in out DATA.SUB_DATA;
                      NEW_ERROR   : in out ERROR_RECORD;
                      NEW_TURN    : in out DATA.TURN_RECORD;
                      ITSTEP,
                      KGTURN      : in INTEGER) is

  K,                        --Array index for distance
  J          : INTEGER;     --Array index for speed
  AMNL,                     --Ambient noise level
  B2,                       --Holds a bearing
  BN1,
  BN2,                      --Background noises
  BNNL,                     --Total background noise
  BR2,                      --Bearing rate
  BT1,
  BT2,                      --Bearing drifts
  PL,                       --Prop loss
  SE,                       --Signal excess
  SL,                       --Ship's sound level
  SN,                       --Self-noise
  DCHK,                     --Used in lost contact determination
  XXX,                      --Random number
  UFREQ       : FLOAT;                  --Holds frequency value
  FREQ        : ENV_DATA.FREQ_ARRAY        := ENV_DATA.NEW_FREQS;
  AMB_NOISE   : ENV_DATA.NOISE_ARRAY       := ENV_DATA.AMBIENT_NOISE;
  USLF        : ENV_DATA.SUB_NOISE_RECORD  := ENV_DATA.NEW_SUB_NOISE;
  VSLF        : ENV_DATA.SHIP_NOISE_RECORD := ENV_DATA.NEW_SHIP_NOISE;
  PLF         : ENV_DATA.PROP_LOSS         := ENV_DATA.NEW_PROP_LOSS;

begin
  --Calculates the bearings between ship and sub and vice versa
  BRGCLC(SUB_RECORD.XU, SUB_RECORD.YU, SHIP_RECORD.XV, SHIP_RECORD.YV,
         SUB_RECORD.CTCBRG);
  BRGCLC(SHIP_RECORD.XV, SHIP_RECORD.YV, SUB_RECORD.XU, SUB_RECORD.YU,
         SHIP_RECORD.CTCBRG);

  --For the two frequencies the sub is searching at (400.0 Hz, 1200.0 Hz)
```

```
for KA in 1..2 loop
  UFREQ := FREQ(KA);
  --determining ambient noise level for this freq and sub depth
  --400.0 Hz
  if UFREQ = 400.0 then
    AMNL := AMB_NOISE(2, SUB_RECORD.KD);
  --1200.0 Hz
  else
    AMNL := AMB_NOISE(3, SUB_RECORD.KD);
  end if;

  --Determining the sub's self-noise, based on it's speed
  J := INTEGER(SUB_RECORD.USPD/5.0);
  if J > 7 then
    J := 7;
  elsif J = 0 then
    J := 1;
  else
    null;
  end if;
  SN := USLF.USN(J);

  --Determine background noise
  BN1 := AMNL - SUB_RECORD.UDI;
  BN2 := SN - SUB_RECORD.UDI;
  --Use the PWRSUM procedure to combine the two sound levels
  PWRSUM (BN1, BN2, BNNL);

  --Determining propagation loss and source level.  Prop loss figures are in
  --the PLF arrays.  Divide the range by two, which gives the K subscript for
  --the appropriate PL.(The figures are in 2 mile increments.)  The ship's
  --source level is a function of speed, and is found in the VSLF arrays,
  --in 5-knot increments.
  --The J subscript is used to retrieve the appropriate value
  J := INTEGER(SHIP_RECORD.VSPD/5.0);
  --This is the maximum figure in the array
  if J > 7 then
    J := 7;
  elsif J = 0 then
    J := 1;
  end if;
  K := INTEGER(SUB_RECORD.SUB_RANGE/2.0);

  --If the range is greater/less than the max/min figures
  if SUB_RECORD.SUB_RANGE > 50.0 then
    K := 25;
  elsif SUB_RECORD.SUB_RANGE < 2.0 then
    K := 1;
  end if;

  --Depending on the frequency of the sound levels, the tables are entered
  if UFREQ = 400.0 then
    SL := VSLF.VSLF1(J);
    PL := PLF.PLF1(SUB_RECORD.KD, K);
  --1200.0 Hz
  else
    SL := VSLF.VSLF2(J);
    PL := PLF.PLF2(SUB_RECORD.KD, K);
  end if;

  --Calculate the signal excess using the sonar equation
  SE := SL - PL - BNNL - SUB_RECORD.URD;

  --Determine the error to add to the equation.  At greater range, the sub's
```

```
  --corrrection is appropriate; at shorter ranges the ship's is used.
  if SUB_RECORD.SUB_RANGE >= 5.0 then
    --add in the lambda sub correction
    NEW_ERROR.LAMSUM := NEW_ERROR.LAMCME + NEW_ERROR.LAMUE;
  else
    --add in the lambda ship correction
    NEW_ERROR.LAMSUM := NEW_ERROR.LAMCME + NEW_ERROR.LAMVE;
  end if;

  --The final signal excess is determined
  SUB_RECORD.SETOT := SE + NEW_ERROR.LAMSUM;
  --Historical record
  NEW_TURN.TSE (KGTURN, KA) := SUB_RECORD.SETOT;

  --Determine if detection is achieved, which happens with a SETOT > 0
  if SUB_RECORD.SETOT >= 0.0 then
    --The detection flags are set
    SUB_RECORD.UDECT(KA, 1) := 1.0;
    SUB_RECORD.UCTC (KA) := TRUE;
    --The lost contact flag is not set
    SUB_RECORD.LCTCT := 0.0;
  --The detection flags are not set
  else
    SUB_RECORD.UDECT(KA, 1) := 0.0;
    SUB_RECORD.UCTC (KA) := FALSE;
  end if;

  --Check for lost contact
  DCHK := SUB_RECORD.UDECT(KA,2) - SUB_RECORD.UDECT(KA,1);
  --If there was contact the previous turn, but not now
  if DCHK > 0.0 then
    SUB_RECORD.ULCTCF := TRUE;
  --Either there has been no change in a no contact situation, or contact
  --is presently held
  else
    SUB_RECORD.ULCTCF := FALSE;
  end if;

  SUB_RECORD.UDECT (KA, 2) := SUB_RECORD.UDECT (KA, 1);

  --Assign the bearing to the frequency if contact is held
  if SUB_RECORD.UCTC(KA) = TRUE then
    SUB_RECORD.UCTCBR (KA) := SUB_RECORD.CTCBRG;
  end if;

end loop;    --KA

--Calculate the bearing drift
SUB_RECORD.BT := SUB_RECORD.B2 - SUB_RECORD.CTCBRG;
SUB_RECORD.BD := ABS (SUB_RECORD.BT);
--Calculate the bearing rate
BR2 := (SUB_RECORD.BT)/(FLOAT(ITSTEP));
SUB_RECORD.B2 := SUB_RECORD.CTCBRG;

--Calculate bearing trend.  Left to right and right to left are inconsistent
--trends.  L to L and R to R are consistent.
if (((SUB_RECORD.BT <= 0.0) AND (SUB_RECORD.BT1 <= 0.0)) OR ((SUB_RECORD.BT1 > 0.0)
     AND (SUB_RECORD.BT > 0.0))) then
  SUB_RECORD.BT := 0.0;
else
  SUB_RECORD.BT := 1.0;
end if;

--For use the next time around for comparison
```

```
  SUB_RECORD.BT1 := SUB_RECORD.BT;
  --Calculate bearing rate
  SUB_RECORD.BR := ABS(BR2 - SUB_RECORD.BR1);
  --For use the next time around for comparison
  SUB_RECORD.BR1 := BR2;

  --Determine if visual/radar detection occur
  --Is the sub at periscope depth?
  if SUB_RECORD.ZU = 60.0 then
    --The sub is radiating
    SUB_RECORD.URAD := TRUE;

    --At periscope depth, with a possibility of visual contact
    --Check if contact occurs
    if SUB_RECORD.SUB_RANGE < 10.0 then
      XXX := U_RAND.NEXT;
      --If the random number falls within the probability
      if XXX <= SUB_RECORD.UPVIS then
        --Visual contact occurs
        SUB_RECORD.UDVR := TRUE;
      end if;

      XXX := U_RAND.NEXT;
      --Check for radar contact
      if XXX <= SUB_RECORD.UPRDR then
        SUB_RECORD.UDVR := TRUE;
      end if;
    --If outside visual contact range
    else
      SUB_RECORD.UDVR := FALSE;
    end if;
  else
    SUB_RECORD.URAD := FALSE;    --No possibility of visual contact
    SUB_RECORD.UDVR := FALSE;
  end if;

end SUB_DETECT;



 --Determines the current environment valuated state space
procedure CURRENT_ENV (SUB_RECORD: in out DATA.SUB_DATA;
                       NEW_TURN  : in out DATA.TURN_RECORD;
                       KGTURN    : in INTEGER;
                       THIS_ENV  : in out DATA.ENV_CHOICE) is

  CTCTYP  : STRING (1..2);--Holds type of contact

begin
  --Determining IVC1
  --No contact
  if ((SUB_RECORD.UCTC(1) = FALSE) AND (SUB_RECORD.UCTC(2) = FALSE)) then
    CTCTYP := "NO";
    THIS_ENV.IVC1 := 1;
  else
    --Broadband and narrow band
    if ((SUB_RECORD.UCTC(1) = TRUE) AND (SUB_RECORD.UCTC(2) = TRUE)) then
      CTCTYP := "BN";
      THIS_ENV.IVC1 := 4;
    else
      --Narrowband contact
      if ((SUB_RECORD.UCTC(1) = TRUE) AND (SUB_RECORD.UCTC(2) = FALSE)) then
        CTCTYP := "NB";
        THIS_ENV.IVC1 := 3;
```

```
    --Broadband contact
    else
      CTCTYP := "BB";
      THIS_ENV.IVC1 := 2;
    end if;
  end if;
end if;

--Determining IVC2
--No contact
if ((CTCTYP = "NO") AND (SUB_RECORD.UDE = FALSE) AND (SUB_RECORD.UDVR = FALSE)) then
  SUB_RECORD.VRE := 'U';    --Sub range estimate unknown
  THIS_ENV.IVC2 := 1;
--Narrowband contact with a small SE
elsif ((CTCTYP = "NB") AND (SUB_RECORD.SETOT <  2.0)) then
  --Long-range contact
  SUB_RECORD.VRE := 'F';
  THIS_ENV.IVC2 := 2;
--Narrow or broadband or ESM detection
elsif (CTCTYP = "NB") OR (CTCTYP = "BB") OR (SUB_RECORD.UDE = TRUE) then
  --Large SE
  if SUB_RECORD.SETOT >= 2.0 then
    --Medium range
    SUB_RECORD.VRE := 'M';
    THIS_ENV.IVC2 := 3;
  --Small SE, long-range contact
  else
    SUB_RECORD.VRE := 'F';
    THIS_ENV.IVC2 := 2;
  end if;

--Narrow and broadband or no doppler or sub has radar/visual detection
elsif ((CTCTYP = "BN") OR (SUB_RECORD.DPLRF = 'Z') OR (SUB_RECORD.UDVR = TRUE)
    OR((SUB_RECORD.SUB_RANGE < 8.0) AND ((CTCTYP = "NB")OR (CTCTYP = "BB")))
    OR (SUB_RECORD.SUB_RANGE < 3.0))then
  --Close range
  SUB_RECORD.VRE := 'N';
  THIS_ENV.IVC2 := 4;
--Medium range (the default)
else
  SUB_RECORD.VRE := 'M';
  THIS_ENV.IVC2 := 3;
end if;

--Determination of IVC3
--Up doppler
if (SUB_RECORD.DPLRF = 'U') then
  THIS_ENV.IVC3 := 1;
--Down/no doppler
else
  THIS_ENV.IVC3 := 2;
end if;

--Determination of IVC4
--Small bearing drift
if (SUB_RECORD.BD < 0.5) then
  THIS_ENV.IVC4 := 1;
--Larger bearing drifts
else
  THIS_ENV.IVC4 := 2;
end if;

--Determination of IVC5 (bearing trend)
if (SUB_RECORD.BT = 1.0) then
```

```
    THIS_ENV.IVC5 := 1;
  else
    THIS_ENV.IVC5 := 2;
  end if;

  --Determination of IVC6 (bearing rate)
  --Small
  if SUB_RECORD.BR < 1.0 then
    THIS_ENV.IVC6 := 1;
  --Larger
  else
    THIS_ENV.IVC6 := 2;
  end if;

  --Determination of IVC7 (signal excess)
  --Large SE
  if SUB_RECORD.SETOT >= 2.0 then
    THIS_ENV.IVC7 := 1;
  --Small SE
  else
    THIS_ENV.IVC7 := 2;
  end if;

  --Data collection
  NEW_TURN.MVC1 (KGTURN) := THIS_ENV.IVC1;
  NEW_TURN.MVC2 (KGTURN) := THIS_ENV.IVC2;
  NEW_TURN.MVC3 (KGTURN) := THIS_ENV.IVC3;
  NEW_TURN.MVC4 (KGTURN) := THIS_ENV.IVC4;
  NEW_TURN.MVC5 (KGTURN) := THIS_ENV.IVC5;
  NEW_TURN.MVC6 (KGTURN) := THIS_ENV.IVC6;
  NEW_TURN.MVC7 (KGTURN) := THIS_ENV.IVC7;

end CURRENT_ENV;


--This determines whether or not the ship will gain contact on the sub
procedure SHIP_DETECT (F2          : out ENV_DATA.FREQ_ARRAY;
                       SHIP_RECORD : in out DATA.SHIP_DATA;
                       SUB_RECORD  : in out DATA.SUB_DATA;
                       NEW_ERROR   : in out ERROR_RECORD) is

  VFREQ : FLOAT;           --Holds frequency value
  AMNL  : FLOAT;           --Holds ambient noise value
  J,                       --Holds ship's speed value for array index
  K     : INTEGER;         --Array index for ship-to-sub distance
  XXX,                     --Holds random number
  SE,
  SL,
  PL,
  SN,
  BN1,
  BN2,
  BNNL,                      --Hold various decibel levels for sonar equation
  DCHK        : FLOAT;       --Used in lost contact determination
  C : FLOAT   := 1530.0;     --Used in determining actual freqs after doppler shift
  AMB_NOISE   : ENV_DATA.NOISE_ARRAY := ENV_DATA.AMBIENT_NOISE;
  VSLF        : ENV_DATA.SHIP_NOISE_RECORD := ENV_DATA.NEW_SHIP_NOISE;
  USLF        : ENV_DATA.SUB_NOISE_RECORD := ENV_DATA.NEW_SUB_NOISE;
  PLF         : ENV_DATA.PROP_LOSS := ENV_DATA.NEW_PROP_LOSS;
  SUB_FREQ    : ENV_DATA.FREQ_ARRAY := ENV_DATA.NEW_FREQS;

begin

  --Checking the frequencies the ship is searching (300.0 Hz, 1200.0 Hz)
```

```
for KA in 3..4 loop
  VFREQ := SUB_FREQ(KA);
  --determining ambient noise level based on frequency and towed body depth
  if VFREQ = 300.0 then
    AMNL := AMB_NOISE(1, SHIP_RECORD.KH);
  --1200.0 Hz
  else
    AMNL := AMB_NOISE(3, SHIP_RECORD.KH);
  end if;

  --Determining the ship's self-noise based on speed
  J := INTEGER(SHIP_RECORD.VSPD/5.0);
  if J > 7 then
    J := 7;
  elsif J < 1 then
    J := 1;
  end if;
  SN := VSLF.VSN(J);

  --Determine background noise by subtracting the differential
  BN1 := AMNL - SHIP_RECORD.VDI;
  BN2 := SN   - SHIP_RECORD.VDI;
  --Use the PWRSUM procedure to combine the two sound levels
  PWRSUM (BN1, BN2, BNNL);

  --Determining propagation loss and source level.  Prop loss figures are in the
  --PLF arrays.  Divide the range by two, which gives the K subscript for the
  --appropriate PL.(The figures are in 2 mile increments.)  The ship's source level
  --is a function of speed, and is found in the VSLF arrays, in 5-knot increments
  --The J subscript is used to retrieve the appropriate value
  J := INTEGER(SUB_RECORD.USPD/5.0);
  if J > 7 then
    J := 7;         --This is the maximum figure in the array
  elsif J = 0 then
    J := 1;
  else
    null;
  end if;

  --If the range is greater/less than the max/min figures
  if SHIP_RECORD.SHIP_RANGE > 50.0 then
    K := 25;
  elsif SHIP_RECORD.SHIP_RANGE < 2.0 then
    K := 1;
  else
    K := INTEGER(SHIP_RECORD.SHIP_RANGE/2.0);
  end if;

  --Depending on the frequency of the sound levels, the tables are entered
  if VFREQ = 300.0 then
    SL := USLF.USLF3(J);
    PL := PLF.PLF3(SHIP_RECORD.KH, SUB_RECORD.KD, K);
  --1200.0 Hz
  else
    SL := USLF.USLF4(J);
    PL := PLF.PLF4(SHIP_RECORD.KH, SUB_RECORD.KD, K);
  end if;

  --Calculate the signal excess
  SE := SL - PL - BNNL - SHIP_RECORD.VRD;

  --Determine the error to add to the equation
  --add in the lambda sub correction
  if SHIP_RECORD.SHIP_RANGE >= 5.0 then
```

```
      NEW_ERROR.LAMSUM := NEW_ERROR.LAMCME + NEW_ERROR.LAMUE;
    --add in the lambda ship correction
    else
      NEW_ERROR.LAMSUM := NEW_ERROR.LAMCME + NEW_ERROR.LAMVE;
    end if;

    --Final signal excess
    SHIP_RECORD.SETOT := SE + NEW_ERROR.LAMSUM;

    --Determine if detection is achieved, which happens with a SETOT > 0
    if SHIP_RECORD.SETOT >= 0.0 then
      --Set the contact flags (Use KA-2 since switching from a four row to
      --a two row array)
      SHIP_RECORD.VDECT(KA-2, 1) := 1.0;
      SHIP_RECORD.VCTC (KA-2)    := TRUE;
      --Determine the actual contact frequency, after adjusting for doppler
      F2 (KA-2) := ((SUB_RECORD.DPLR/C) * SUB_FREQ(KA)) + SUB_FREQ(KA);

    --Set the flags for no contact
    else
      SHIP_RECORD.VDECT(KA-2, 1) := 0.0;
      SHIP_RECORD.VCTC (KA-2) := FALSE;
    end if;

    --Check for lost contact by checking for the difference in the flags from
    --one turn to the next
    DCHK := SHIP_RECORD.VDECT(KA-2,2) - SHIP_RECORD.VDECT(KA-2,1);
    --Set the lost contact flag
    if DCHK > 0.0 then
      SHIP_RECORD.VLCTCF(KA-2) := TRUE;
    --No change from the last time, or contact has been gained
    else
      SHIP_RECORD.VLCTCF(KA-2) := FALSE;
    end if;
    --Reset the values for use next turn
    SHIP_RECORD.VDECT (KA-2, 2) := SHIP_RECORD.VDECT (KA-2, 1);

    --Calculate the sub's contact bearing if there is contact
    if SHIP_RECORD.VCTC(KA-2) = TRUE then
      SHIP_RECORD.VCTCBR (KA-2) := SHIP_RECORD.CTCBRG;
    end if;

  end loop;   --KA

  --Gives doppler info to user
  if (SHIP_RECORD.VCTC (1) OR SHIP_RECORD.VCTC (2)) then
    if SUB_RECORD.SUB_RANGE < SHIP_RECORD.SHIP_RANGE then
      PUT_LINE ("Contact is up doppler.");
    end if;
  end if;

  SHIP_RECORD.SHIP_RANGE := SUB_RECORD.SUB_RANGE;

  --Determine if visual/radar detection occur
  if SUB_RECORD.URAD = TRUE then
    XXX := U_RAND.NEXT;
    --The sub is at periscope depth
    if XXX <= SHIP_RECORD.VPRDR and SHIP_RECORD.SHIP_RANGE < 8.0 then
      --Radar contact
      SHIP_RECORD.VDVR := TRUE;
      PUT ("Pop-up contact, bearing: ");
      PUT (SHIP_RECORD.CTCBRG, FORE => 3, AFT => 1, EXP => 0);
      PUT ("Range : ");PUT (SHIP_RECORD.SHIP_RANGE, FORE => 3, AFT => 1, EXP => 0);
      NEW_LINE;
```

```
    end if;

    XXX := U_RAND.NEXT;
    --The probability and range requirements for visual contact are met
    if XXX <= SHIP_RECORD.VPVIS  AND (SHIP_RECORD.SHIP_RANGE <= 3.0) then
      SHIP_RECORD.VDVR := TRUE;
      SET_COL (48);
      PUT_LINE("Ship in contact visually.  Bearing : ");
      PUT (SUB_RECORD.CTCBRG, FORE => 3, AFT => 1, EXP => 0); NEW_LINE;
      SET_COL(75);
      PUT ("Estimated Range: ");
      XXX := U_RAND.NEXT;

      --Adds error into determination of visual contact range
      if XXX < 0.50 then
        XXX := SUB_RECORD.SUB_RANGE + XXX;
      else
        XXX := SUB_RECORD.SUB_RANGE - XXX;
      end if;

      if XXX < 0.2 then
        XXX := 0.5;
      end if;

      PUT (XXX, FORE => 2, AFT => 1, EXP => 0);
      NEW_LINE;
    end if;
  --If outside visual contact range
  else
    --No contact
    SHIP_RECORD.VDVR := FALSE;
  end if;

  --Sub is radiating and within range
  if ((SUB_RECORD.URAD = TRUE) AND (SHIP_RECORD.SHIP_RANGE <= 12.0)) then
    XXX := U_RAND.NEXT;

    --XXX less than the prob of esm detection, then detection occurs
    if (XXX <= SHIP_RECORD.VPEWD) then
      SHIP_RECORD.VDE := TRUE;
    end if;

  --No contact
  else
    SHIP_RECORD.VDE := FALSE;
  end if;

end SHIP_DETECT;


--Outputs detection results
procedure DETECT_RESULTS (SHIP_RECORD : in out DATA.SHIP_DATA;
                          SUB_RECORD  : in out DATA.SUB_DATA;
                          TIMER       : in DATA.TIME_RECORD;
                          F2          : in ENV_DATA.FREQ_ARRAY;
                          PLAY        : in out BOOLEAN) is

  CHOICE : BOOLEAN := TRUE;
  STOP   : INTEGER;
  FREQ   : ENV_DATA.FREQ_ARRAY := ENV_DATA.NEW_FREQS;

begin

  for KA in 1..2 loop
```

```
  if (SHIP_RECORD.VCTC (KA) = TRUE) then

    --If the ship gained contact or held an old contact
    if SHIP_RECORD.VLCTCF (KA) /= TRUE then
      PUT_LINE ("Ship gained contact ");
      PUT (F2(KA),FORE => 5, AFT => 1, EXP => 0);
      PUT (" HZ, BRG: ");
      PUT (SHIP_RECORD.VCTCBR(KA),FORE => 5, AFT => 1, EXP => 0);
      NEW_LINE;
      SHIP_RECORD.VLCTCF (KA) := TRUE;
    end if;

  --Contact was lost this turn on this frequency
  else
    if (SHIP_RECORD.VLCTCF (KA) = TRUE) then
      PUT_LINE ("Ship lost contact ");
      PUT(F2(KA),FORE => 5, AFT => 1, EXP => 0);
      PUT("HZ, BRG: ");
      PUT (SHIP_RECORD.VCTCBR(KA), FORE => 5, AFT => 1, EXP => 0);
      SHIP_RECORD.VLCTCF (KA) := FALSE;
    end if;
  end if;
end loop;                        --KA


--ESM contact
if SHIP_RECORD.VDE = TRUE then
  SET_COL (48);
  PUT("Snoop Series - Brg: ");
  PUT(SHIP_RECORD.CTCBRG, FORE => 5, AFT => 1, EXP => 0);
  NEW_LINE;
  --Reset the flag
  SHIP_RECORD.VDE := FALSE;
end if;

--Current update, and lets the user end the simulation if desired
PUT_LINE ("Contact Update: ");
PUT ("Gametime : ");
SET_COL(12);
PUT ("DAY: ");
PUT (TIMER.GDAY, WIDTH => 2);
SET_COL (21);
PUT ("HOUR: ");
PUT (TIMER.GHR, WIDTH => 2);
SET_COL (32);
PUT ("MINUTES: ");
PUT (TIMER.GMIN, WIDTH => 2);
NEW_LINE;

--Checks each freq for contact and outputs the data if there is contact
for KA in 1..2 loop
  if SHIP_RECORD.VCTC(KA) = TRUE then
    PUT_LINE ("Ship in contact: ");
    PUT ("Bearing: ");
    PUT(SHIP_RECORD.VCTCBR (KA), FORE => 5, AFT => 1, EXP => 0);
    PUT(F2(KA),FORE => 5, AFT => 1, EXP => 0);
    PUT(" Hz ");
    NEW_LINE;
  else
    PUT ("Not in Contact, Freq ");
    PUT(FREQ(KA + 2), FORE => 5, AFT => 1, EXP => 0);
    PUT (" Hz.");
    NEW_LINE;
  end if;
```

```
   end loop;

   --Determine if the user wishes to continue
   while CHOICE loop
     PUT_LINE ("Enter 1 to exit or 2 to continue.");
     GET (STOP);
     NEW_LINE;

     if STOP = 2 then
       PLAY := TRUE;
       CHOICE := FALSE;
     elsif STOP = 1 then
       PLAY := FALSE;
       CHOICE := FALSE;
     else
       PUT_LINE ("Error in choice.  Please try again.");
     end if;
   end loop;

 end DETECT_RESULTS;



 --Runs the above procedures
 procedure RUN_DETECTION ( IELAPT,
                           ITSTEP,
                           KGTURN      : in INTEGER;
                           SHIP_RECORD : in out DATA.SHIP_DATA;
                           SUB_RECORD  : in out DATA.SUB_DATA;
                           NEW_ERROR   : in out ERROR_RECORD;
                           NEW_TURN    : in out DATA.TURN_RECORD;
                           F2          : in out ENV_DATA.FREQ_ARRAY;
                           TIMER       : in DATA.TIME_RECORD;
                           THIS_ENV    : in out DATA.ENV_CHOICE;
                           PLAY        : in out BOOLEAN) is


 begin
   --Determine the error levels for this turn
   DETECT_VARIABLES (IELAPT,SHIP_RECORD, SUB_RECORD, NEW_ERROR, NEW_TURN, KGTURN);

   --Determine if the sub holds contact on the ship
   SUB_DETECT (SHIP_RECORD, SUB_RECORD,NEW_ERROR,NEW_TURN,ITSTEP, KGTURN);

   --Determine what the current environment is
   CURRENT_ENV (SUB_RECORD, NEW_TURN, KGTURN, THIS_ENV);

   --Determine if the ship holds contact on the sub
   SHIP_DETECT (F2, SHIP_RECORD, SUB_RECORD,NEW_ERROR);

   --Output the results
   DETECT_RESULTS (SHIP_RECORD, SUB_RECORD, TIMER, F2, PLAY);

 end RUN_DETECTION;

nd DETECTION;
```

```
--Title:        decide_s.a
--Subject:      Ship updates course, speed, array depth and decides
--              whether or not to attack.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

with DATA, GENALG, INITIALIZE, DETECTION;


package DECIDE is

  --Ship decides what changes to make, and whether or not to attack
  procedure SHIP_PLAY (SHIP_RECORD : in out DATA.SHIP_DATA;
                       SUB_RECORD  : in out DATA.SUB_DATA;
                       PROB        : in DATA.PROB_RANGE;
                       HIT         : in out BOOLEAN);



  --Ship conducts attack
  procedure SHIP_ATTACK (SHIP_RECORD : in out DATA.SHIP_DATA;
                         SUB_RECORD  : in out DATA.SUB_DATA;
                         PROB        : in DATA.PROB_RANGE;
                         HIT         : out BOOLEAN);


  --If in contact, use the genetic algorithm.  Otherwise, use one of the
  --lost contact procedures
  procedure SUB_CONTACT (SUB_RECORD : in out DATA.SUB_DATA;
                         SHIP_RECORD: in out DATA.SHIP_DATA;
                         NEW_TURN   : in out DATA.TURN_RECORD;
                         PROB       : in DATA.PROB_RANGE;
                         KGTURN,
                         KE         : in INTEGER;
                         KGPOOL     : in INITIALIZE.BIT_STRINGS;
                         MOVE       : in INITIALIZE.MOVE_NUM;
                         THIS_ENV   : in DATA.ENV_CHOICE;
                         SUNK       : out BOOLEAN);



  --The sub tries to attack, which is not allowed if not within range
  procedure SUB_ATTACK (SUB_RECORD    : in out DATA.SUB_DATA;
                        SHIP_RECORD   : in out DATA.SHIP_DATA;
                        PROB          : in DATA.PROB_RANGE;
                        SUNK          : out BOOLEAN);



  --The sub is out of range for an attack, so it conducts an approaching movement.
  procedure SUB_APPROACH (SUB_RECORD : in out DATA.SUB_DATA);


  --SUB's lost contact search procedure
  procedure LOST_CONTACT (SUB_RECORD : in out DATA.SUB_DATA);


  --Sub's random search procedure
  procedure RANDOM_SEARCH(SUB_RECORD : in out DATA.SUB_DATA);


  --Runs the ship/sub procedures for determining moves
  procedure RUN_DECIDE (SUB_RECORD : in out DATA.SUB_DATA;
                        SHIP_RECORD: in out DATA.SHIP_DATA;
```

```
                    NEW_TURN    : in out DATA.TURN_RECORD;
                    PROB        : in DATA.PROB_RANGE;
                    KGTURN      : in INTEGER;
                    KGPOOL      : in out INITIALIZE.BIT_STRINGS;
                    MOVE        : in out INITIALIZE.MOVE_NUM;
                    THIS_ENV    : in out DATA.ENV_CHOICE;
                    SUB_HIT     : in out BOOLEAN;
                    ENV         : in out INITIALIZE.ENV_NUM;
                    DEC         : in out GENALG.DECI_FIT;
                    WMOVE       : in out INITIALIZE.MOVE_VALUE;
                    WENV        : in out INITIALIZE.ENV_VALUE;
                    NUM_STRINGS: in INTEGER;
                    FIVE_ENV    : in out GENALG.OLD_ENV;
                    SUNK        : out BOOLEAN);

end DECIDE;
```

```
--Title:        decide_b.a
--Subject:      Ship updates course, speed, array depth and decides
--              whether or not to attack.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

with TEXT_IO, MATH_LIB, U_RAND;
use TEXT_IO, MATH_LIB;

package body DECIDE is

  package INTEGER_INOUT is new INTEGER_IO (INTEGER);
  package FLOAT_INOUT is new FLOAT_IO (FLOAT);

  use INTEGER_INOUT, FLOAT_INOUT;

  --Determines ship damage following a torpedo hit
  procedure VDMG (SHIP_RECORD : in out DATA.SHIP_DATA;
                  BTSHOT,
                  PROB    : in DATA.PROB_RANGE;
                  SUNK    : out BOOLEAN) is

    BTATK  : FLOAT := BTSHOT - 180.0; --bearing of the attack from ship
    RB     : FLOAT;                   --Relative bearing


    --Amidships hit
    procedure AMIDHIT (PMSUNK  : in FLOAT;
                       VSPD    : out FLOAT;
                       SUNK    : out BOOLEAN) is

      XXX : FLOAT;

    begin
      XXX := U_RAND.NEXT;
      --If the random number falls within the prob of sinking
      if XXX <= PMSUNK then
        PUT_LINE ("Ship is hit amidships, sinks.");
        SUNK := TRUE;
      else
        PUT_LINE ("Ship is hit amidships, remains afloat.");
        --Ship loses speed
        VSPD := 0.0;
        SUNK := FALSE;
      end if;
    end AMIDHIT;


    --Torpedo hit aft
    procedure AFTHIT (PESUNK  : in FLOAT;
                      VSPD    : out FLOAT;
                      SUNK    : out BOOLEAN) is

      XXX : FLOAT;

    begin
      XXX := U_RAND.NEXT;
      --If the random number falls within the probability
      if XXX <= PESUNK then
        PUT_LINE ("Ship is hit aft, sinks.");
        SUNK := TRUE;
      else
        PUT_LINE ("Ship is hit aft, remains afloat.");
        --Ship loses speed
```

```
    VSPD := 0.0;
    SUNK := FALSE;
  end if;
end AFTHIT;


--Torpedo hit forward
procedure FORHIT (PESUNK  : in FLOAT;
                  VSPD    : out FLOAT;
                  SUNK    : out BOOLEAN) is

 XXX : FLOAT;

begin
  XXX := U_RAND.NEXT;
  --If the random number falls within the probability
  if XXX <= PESUNK then
    PUT_LINE ("Ship is hit forward, sinks.");
    SUNK := TRUE;
  else
    PUT_LINE ("Ship is hit forward, can maintain steerageway.");
    --Ship loses speed
    VSPD := 3.0;     --Speed reduced to three knots.
    SUNK := FALSE;
  end if;
end FORHIT;


--Begin VDMG
begin
  --BTATK is the bearing of the torpedo attack
  if BTATK < 0.0 then
    BTATK := BTATK + 360.0;
  elsif BTATK > 360.0 then
    BTATK := BTATK - 360.0;
  end if;

  --determine relative bearing of the attack
  if SHIP_RECORD.VCSE >= BTATK then
    RB := 360.0 - (SHIP_RECORD.VCSE - BTATK);
  else
    RB := BTATK - SHIP_RECORD.VCSE;
  end if;

  --Based on the relative bearing of the attack, determine where the
  --torpedo will hit, and call the appropriate sub-procedure
  if (RB >= 60.0) AND (RB <= 120.0) then
    AMIDHIT ( PROB.PMSUNK, SHIP_RECORD.VSPD, SUNK);
  elsif (RB >= 240.0) AND (RB <= 300.0) then
    AMIDHIT ( PROB.PMSUNK, SHIP_RECORD.VSPD, SUNK);
  elsif (RB >= 120.0) AND (RB <= 240.0) then
    AFTHIT ( PROB.PESUNK, SHIP_RECORD.VSPD, SUNK);
  else
    FORHIT ( PROB.PESUNK, SHIP_RECORD.VSPD, SUNK);
  end if;
  PUT_LINE ("DECIDE.VDMG 4 ");

end VDMG;


--Ship conducts attack
--The user will input the desired weapon entry point.  This will be
--compared to the sub's position, and the prob of hit will be applied.
--Two criteria will be applied:  The firing unit must be in contact and
```

```
--the target must be within firing range.
procedure SHIP_ATTACK (SHIP_RECORD : in out DATA.SHIP_DATA;
                       SUB_RECORD  : in out DATA.SUB_DATA;
                       PROB        : in DATA.PROB_RANGE;
                       HIT         : out BOOLEAN) is      --passed in FALSE

  RWEP,          --Range from weapon to sub
  BWEP,          --Weapon's bearing
  XWEP,          --Range in the X axis
  YWEP,          --Range in the Y axis
  X,             --Holds X-axis range difference
  Y,             --Holds Y-axis range differential
  XXX          : FLOAT;     --Random number
  MY_PI        : FLOAT := 3.14159;

begin
  HIT := FALSE;
  PUT_LINE ("Enter weapon entry point, Range (in nm) and bearing (in");
  PUT_LINE ("Degrees.  Please use this format 012.5 345.3");
  GET (RWEP);
  GET (BWEP);
  SKIP_LINE;

  --Convert weapons angle to to radians
  BWEP := (BWEP/180.0) * MY_PI;

  --Convert to X and Y components
  XWEP := SHIP_RECORD.XV + RWEP * MATH_LIB.SIN(BWEP);
  YWEP := SHIP_RECORD.YV + RWEP * MATH_LIB.COS (BWEP);

  --The difference in distance between sub and entry point
  X := XWEP - SUB_RECORD.XU;
  Y := YWEP - SUB_RECORD.YU;
  RWEP := MATH_LIB.SQRT ((X**2) + (Y**2));
  DETECTION.BRGCLC (XWEP, YWEP, SUB_RECORD.XU, SUB_RECORD.YU, BWEP);
  SHIP_RECORD.TORPAU := TRUE;

  --The sub is within range of the torpedo
  if RWEP <= PROB.VTPRG then

      --Roll the dice for a hit
      XXX := U_RAND.NEXT;
      if XXX <= PROB.PHIT then
        PUT_LINE ("Sub has been hit.");        --Exit the program
        HIT := TRUE;
      else
        PUT_LINE ("Torpedo missed.");
        SHIP_RECORD.TORPAU := FALSE;
      end if;
  --Not within range
  else
    PUT_LINE ("Not within range.");
  end if;

end SHIP_ATTACK;

--Ship decides what changes to make, and whether or not to attack
procedure SHIP_PLAY (SHIP_RECORD : in out DATA.SHIP_DATA;
                     SUB_RECORD  : in out DATA.SUB_DATA;
                     PROB        : in DATA.PROB_RANGE;
                     HIT         : in out BOOLEAN) is

  CHANGE,
  CHOICE  : CHARACTER;
```

```
  XVSTEP,
  YVSTEP  : FLOAT;--Position change in X and Y axes
  XXX     : FLOAT;--Random number
  NO_ERRORS    : BOOLEAN := TRUE;

begin
  PUT_LINE ("The current ship information is:");
  SET_COL (45);
  PUT ("Ship Posit: X NM = ");
  PUT(SHIP_RECORD.XV, FORE => 5, AFT => 2, EXP => 0);
  NEW_LINE;
  SET_COL (58);
  PUT("Y NM = ");
  PUT(SHIP_RECORD.YV, FORE => 5, AFT => 2, EXP => 0);
  NEW_LINE(2);
  SET_COL (45);
  PUT ("Course: ");
  SET_COL (58);
  PUT(SHIP_RECORD.VCSE, FORE => 5, AFT => 2, EXP => 0);
  NEW_LINE;
  SET_COL (45);
  PUT ("Speed: ");
  SET_COL (58);
  PUT (SHIP_RECORD.VSPD, FORE => 5, AFT => 2, EXP => 0);
  NEW_LINE;
  SET_COL (45);
  PUT ("Array Depth: ");
  SET_COL (58);
  PUT (SHIP_RECORD.ZV, FORE => 5, AFT => 2, EXP => 0);
  NEW_LINE;
  PUT ("Engagement Status: ");
  if SUB_RECORD.TORPAV = TRUE then
    PUT_LINE ("Hot.");
  else
    PUT_LINE ("Cold.");
  end if;
  NEW_LINE (2);

  while NO_ERRORS loop

    NO_ERRORS := FALSE;

    PUT_LINE ("Any changes? 1-Yes, 2-No");
    GET (CHANGE);
    SKIP_LINE;

    if CHANGE = '1' then

      --Select which entries will be changed
      while CHANGE = '1' loop
        PUT_LINE ("Select 1-Course, 2-Speed, 3-Array depth, 4-Attack");
        GET (CHOICE);
        SKIP_LINE;

        --Block for exception handler
        begin

          if CHOICE = '1' then
            PUT_LINE ("Enter new course: XXX.X degrees true");
            GET (SHIP_RECORD.VCSE);
            SKIP_LINE;
          elsif CHOICE = '2' then
            PUT_LINE ("Enter new speed: XX.X knots (Max of 29.0)");
            GET (SHIP_RECORD.VSPD);
```

```
            SKIP_LINE;
          elsif CHOICE = '3' then
            PUT_LINE ("Enter array depth: 90.0, 150.0,300.0,450.0 feet");
            GET(SHIP_RECORD.ZV);
            SKIP_LINE;
          elsif CHOICE = '4' then
            SHIP_RECORD.TORPAU := TRUE;
          else
            PUT_LINE ("That wasn't a choice.");
          end if;

        exception
          when DATA_ERROR =>
            PUT_LINE ("Improper format.  Please try again.");
            NO_ERRORS := TRUE;
        end;

        PUT_LINE ("Any more changes? If yes, enter 1.  If no, enter 2");
        GET(CHOICE);
        SKIP_LINE;

        if CHOICE = '2' then
          CHANGE := '2';
        --Stay in the loop
        elsif CHOICE = '1' then
          CHANGE := '1';
        else
          PUT_LINE ("That was not an option.  Try again.");
          CHANGE := '1';
        end if;

      end loop; --CHANGE = 1

      --If contact is held, and attack is chosen, the attack is executed
      if ((SHIP_RECORD.VCTC(1) = TRUE) OR (SHIP_RECORD.VCTC(2) = TRUE)
        OR (SHIP_RECORD.VDVR = TRUE)  OR (SHIP_RECORD.VDE = TRUE)) then
        --The ship is attacking
        if SHIP_RECORD.TORPAU = TRUE then
          SHIP_ATTACK (SHIP_RECORD,  SUB_RECORD, PROB, HIT);
        end if;
      else
        PUT_LINE ("Not in contact.  Cannot shoot.");
      end if;

    else
      PUT_LINE ("Improper entry.  Please try again.");
      NO_ERRORS := TRUE;
    end if;             --CHANGE = 1

  end loop;             --NO_ERRORS

end SHIP_PLAY;

--Sub attempts an attack.  The firing unit must be in contact and the target
--must be within range.
procedure SUB_ATTACK (SUB_RECORD  : in out DATA.SUB_DATA;
                      SHIP_RECORD : in out DATA.SHIP_DATA;
                      PROB        : in DATA.PROB_RANGE;
                      SUNK        : out BOOLEAN) is

  XXX : FLOAT;
  XUSTEP,
  YUSTEP : FLOAT;
```

```
begin

  SUNK := FALSE;
  --Determine the bearing between the sub and the target
  --Target is within range
  if SUB_RECORD.SUB_RANGE <= PROB.UTPRG then
    --If the sub holds contact
    if ((SUB_RECORD.UCTC(1) = TRUE) OR (SUB_RECORD.UCTC(2) = TRUE)
        OR (SUB_RECORD.UDVR = TRUE) OR (SUB_RECORD.UDE = TRUE)) then
      --The attack is launched
      XXX := U_RAND.NEXT;
      --If XXX falls within the hit prob
      if XXX <= PROB.PHIT then
        PUT_LINE ("Ship has been hit.");
        --Determine the damage
        VDMG (SHIP_RECORD, SUB_RECORD.BTSHOT,  PROB, SUNK);
        SUNK := TRUE;

      --The torpedo misses
      else
        PUT_LINE ("Torpedo missed.");
        --Take evasive action
        SUB_RECORD.ZU := 600.0;
        SUB_RECORD.USPD := 22.0;
        --Evasive course
        if SUB_RECORD.BTSHOT > 180.0 then
          SUB_RECORD.UCSE := SUB_RECORD.BTSHOT - 180.0;
        else
          SUB_RECORD.UCSE := SUB_RECORD.BTSHOT + 180.0;
        end if;
      end if;
    --If the sub does not hold contact
    end if;

  end if;

end SUB_ATTACK;

--The contact is out of attack range, so the sub conducts a closing
--movement
procedure SUB_APPROACH (SUB_RECORD : in out DATA.SUB_DATA) is

  UC       : FLOAT;
  UCC,
  XUSTEP,
  YUSTEP   : FLOAT;

begin
  --Determine course correction based on the bearing trend.  UCC will be used
  --to determine if the course change should be left or right
  if SUB_RECORD.BT < 0.0 then
    UCC := -1.0;
  else
    UCC := 1.0;
  end if;

  --Determine approach course
  SUB_RECORD.UCSE := SUB_RECORD.CTCBRG + (UCC * SUB_RECORD.DUCSE);

  --For courses greater than 360/less than 0
  if SUB_RECORD.UCSE > 360.0 then
    SUB_RECORD.UCSE := SUB_RECORD.UCSE - 360.0;
  elsif SUB_RECORD.UCSE < 0.0 then
    SUB_RECORD.UCSE := SUB_RECORD.UCSE + 360.0;
```

```
  end if;

end SUB_APPROACH;


--The lost contact search procedure
procedure LOST_CONTACT(SUB_RECORD : in out DATA.SUB_DATA) is

  XUSTEP,
  YUSTEP    : FLOAT;

begin
  --If no contact
  if SUB_RECORD.LCTCT = 0.0 then
    --New course
    SUB_RECORD.UCSE := SUB_RECORD.UCSE + 45.0;
      --Replace heading with actual, if > 360
      if SUB_RECORD.UCSE > 360.0 then
        SUB_RECORD.UCSE := SUB_RECORD.UCSE - 360.0;
      end if;

  else
    if SUB_RECORD.UCSE < 0.0 then
      SUB_RECORD.UCSE := SUB_RECORD.UCSE + 360.0;
    end if;

    for JT in 1..3 loop
      if (INTEGER(SUB_RECORD.LCTCT) = JT) then
        SUB_RECORD.UCSE := SUB_RECORD.UCSE - 90.0;
        exit;
      end if;
    end loop;
  end if;
  --Update the lost contact counter
  SUB_RECORD.LCTCT := 1.0 + SUB_RECORD.LCTCT;

end LOST_CONTACT;


--Random search procedure
procedure RANDOM_SEARCH (SUB_RECORD : in out DATA.SUB_DATA)is

  XXX,
  XUSTEP,
  YUSTEP    : FLOAT;

begin
  XXX := U_RAND.NEXT;
  --New, random course
  if XXX <= 0.2 then
    SUB_RECORD.UCSE := XXX * 360.0;
  --Stay the old course
  else
    null;
  end if;

  --New speed = patrol speed
  SUB_RECORD.USPD := 15.0;
  --Update positions

  XXX := U_RAND.NEXT;
  --New sub depth
  if XXX <= 0.05 then
    SUB_RECORD.ZU := 60.0;
```

```
  elsif XXX <= 0.40 then
    SUB_RECORD.ZU := 150.0;
  elsif XXX <= 0.80 then
    SUB_RECORD.ZU := 300.0;
  elsif XXX <= 0.90 then
    SUB_RECORD.ZU := 450.0;
  else
    SUB_RECORD.ZU := 600.0;
  end if;

end RANDOM_SEARCH;


--If in contact, use the genetic algorithm.  Otherwise, use one of the
--lost contact procedures.
procedure SUB_CONTACT (SUB_RECORD : in out DATA.SUB_DATA;
                       SHIP_RECORD: in out DATA.SHIP_DATA;
                       NEW_TURN   : in out DATA.TURN_RECORD;
                       PROB       : in DATA.PROB_RANGE;
                       KGTURN,
                       KE         : in INTEGER;
                       KGPOOL     : in INITIALIZE.BIT_STRINGS;
                       MOVE       : in INITIALIZE.MOVE_NUM;
                       THIS_ENV   : in DATA.ENV_CHOICE;
                       SUNK       : out BOOLEAN) is

  IUC1,
  IUC2,
  IUC3,
  IUC4 : INTEGER;    --Hold values for tactical choices

begin
  SUNK := FALSE;
  --Converts KGPOOL index to move matrix index
  for IB in 1..2 loop
    for IC in 1..7 loop
      for ID in 1..6 loop
        for IE in 1..6 loop
          if MOVE (IB, IC, ID, IE) = KGPOOL(1, KE) then
            IUC1 := IB;
            IUC2 := IC;
            IUC3 := ID;
            IUC4 := IE;
          end if;
        end loop;
      end loop;
    end loop;
  end loop;

  --Enters into the historical database
  NEW_TURN.MUC1 (KGTURN) := IUC1;
  NEW_TURN.MUC2 (KGTURN) := IUC2;
  NEW_TURN.MUC3 (KGTURN) := IUC3;
  NEW_TURN.MUC4 (KGTURN) := IUC4;
  NEW_TURN.TMOVE (KGTURN) := MOVE (IUC1, IUC2, IUC3, IUC4);

  if IUC1 = 1 then
    --Sub has decided to attack
    SUB_RECORD.TORPAV := TRUE;
  else
    --No attack
    SUB_RECORD.TORPAV := FALSE;
  end if;
```

```
  --Determine the course adjustment
  if (IUC2 = 1) then
    SUB_RECORD.DUCSE := 45.0;
  elsif (IUC2 = 2) then
    SUB_RECORD.DUCSE := 30.0;
  elsif (IUC2 = 3) then
    SUB_RECORD.DUCSE := 60.0;
  elsif (IUC2 = 4) then
    SUB_RECORD.DUCSE := 0.0;
  elsif (IUC2 = 5) then
    SUB_RECORD.DUCSE := 90.0;
  elsif (IUC2 = 6) then
    SUB_RECORD.DUCSE := -90.0;
  elsif (IUC2 = 7) then
    SUB_RECORD.DUCSE := 180.0;
  else
    PUT_LINE ("ERROR IN IUC2");
  end if;

  --Determine the new speed
  if (IUC3 = 1) then
    SUB_RECORD.USPD := 8.0;
  elsif (IUC3 = 2) then
    SUB_RECORD.USPD := 12.0;
  elsif (IUC3 = 3) then
    SUB_RECORD.USPD := 16.0;
  elsif (IUC3 = 4) then
    SUB_RECORD.USPD := 4.0;
  elsif (IUC3 = 5) then
    SUB_RECORD.USPD := 20.0;
  else
    SUB_RECORD.USPD := 24.0;
  end if;

  --Determine the new depth
  if (IUC4 = 1) then
    SUB_RECORD.ZU := 150.0;
  elsif (IUC4 = 2) then
    null;
  elsif (IUC4 = 3) then
    SUB_RECORD.ZU := 60.0;
  elsif (IUC4 = 4) then
    SUB_RECORD.ZU := 300.0;
  elsif (IUC4 = 5) then
    SUB_RECORD.ZU := 450.0;
  else
    SUB_RECORD.ZU := 600.0;
  end if;

  --The sub has decided to attack, the target is "near," and the sub is in
  --contact
  if ((SUB_RECORD.TORPAV = TRUE) AND ((THIS_ENV.IVC2 = 4) OR
       (SUB_RECORD.UDVR = TRUE))) then
    SUB_ATTACK(SUB_RECORD,SHIP_RECORD,PROB, SUNK);
  --Otherwise, continue the approach manuever
  else
    SUB_APPROACH (SUB_RECORD);
  end if;

end SUB_CONTACT;


--Runs the ship/sub procedures for determining moves
procedure RUN_DECIDE(SUB_RECORD : in out DATA.SUB_DATA;
```

```
                    SHIP_RECORD: in out DATA.SHIP_DATA;
                    NEW_TURN   : in out DATA.TURN_RECORD;
                    PROB       : in DATA.PROB_RANGE;
                    KGTURN     : in INTEGER;
                    KGPOOL     : in out INITIALIZE.BIT_STRINGS;
                    MOVE       : in out INITIALIZE.MOVE_NUM;
                    THIS_ENV   : in out DATA.ENV_CHOICE;
                    SUB_HIT    : in out BOOLEAN;
                    ENV        : in out INITIALIZE.ENV_NUM;
                    DEC        : in out GENALG.DECI_FIT;
                    WMOVE      : in out INITIALIZE.MOVE_VALUE;
                    WENV       : in out INITIALIZE.ENV_VALUE;
                    NUM_STRINGS: in INTEGER;
                    FIVE_ENV   : in out GENALG.OLD_ENV;
                    SUNK       : out BOOLEAN) is


  KE : INTEGER;

begin

  SUB_HIT := FALSE;
  SUNK := FALSE;

  --The ship runs its turn
  SHIP_PLAY (SHIP_RECORD,  SUB_RECORD, PROB, SUB_HIT);

  --Until the sub is hit, or the user decides to quit, the simulation will
  --continue
  if (not SUB_HIT) then
    --If the sub is in contact, the genetic algorithm will be used
    if (SUB_RECORD.UCTC(1) = TRUE) OR (SUB_RECORD.UCTC(2) = TRUE)
      OR (SUB_RECORD.UDVR = TRUE) OR (SUB_RECORD.UDE = TRUE) then

      --The genetic algorithm determines the strongest string
      GENALG.DECISION_FIT (DEC, ENV, WMOVE, WENV, KGPOOL, NEW_TURN,
                           THIS_ENV, KGTURN, KE,  FIVE_ENV, NUM_STRINGS);
      --With the updated bit strings, the sub contact procedure is called
      SUB_CONTACT (SUB_RECORD,SHIP_RECORD,NEW_TURN,PROB,KGTURN,KE,
                   KGPOOL,MOVE,THIS_ENV, SUNK);

    --If contact has been lost
    --If contact is not held,
    elsif (SUB_RECORD.ULCTCF = TRUE) then
      NEW_TURN.TENV  (KGTURN) := 316;
      NEW_TURN.TMOVE (KGTURN) := 316;
      --If contact has been lost recently, continue with the lost contact plan
      if (SUB_RECORD.LCTCT <= 8.0) then
        LOST_CONTACT(SUB_RECORD);
      --Otherwise, conduct a random search
      else
        SUB_RECORD.ULCTCF := FALSE;
        RANDOM_SEARCH ( SUB_RECORD);
      end if;
    --If contact has not been lost, and is not held, conduct a random search
    else
      NEW_TURN.TENV  (KGTURN) := 316;
      NEW_TURN.TMOVE (KGTURN) := 316;
      RANDOM_SEARCH ( SUB_RECORD);
    end if;
  else
    NEW_TURN.TENV  (KGTURN) := 316;
    NEW_TURN.TMOVE (KGTURN) := 316;
  end if;
```

```
   end RUN_DECIDE;
end DECIDE;
```

```
--Title       : genalg_s.a
--Subject     : Contains the genetic algorithm procedures for the simulator
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

with INITIALIZE, DATA;


package GENALG is

  type WT_DEC      is array (1..INITIALIZE.NUM_STRINGS)          of FLOAT;
  type DECI_FIT    is array (1..INITIALIZE.NUM_STRINGS, 1..5)    of FLOAT;
  type ENV_ARRAY   is array (1..INITIALIZE.NUM_STRINGS, 1..512) of INTEGER;
  type STR_VALUE   is array (1..INITIALIZE.NUM_STRINGS)          of FLOAT;
  type AVG_VALUE   is array (1..INITIALIZE.NUM_STRINGS)          of FLOAT;
  type SPACE_VALUE is array (1..INITIALIZE.NUM_STRINGS, 1..512) of INTEGER;
  type TEMP_ARRAY  is array (1..512)                    of INTEGER;
  type OLD_ENV     is array (1..5)                      of INTEGER;


  --Calculates the fitness based on the total of each alleles joint space value
  procedure GRADE(ALL_KE                : in OLD_ENV;
                  IC                    : in out INITIALIZE.BIT_STRINGS;
                                         --16x512, value of 1-504
                  ALL_STR_VAL           : out FLOAT;
                  WENV                  : in INITIALIZE.ENV_VALUE;     --1x512
                  WMOVE                 : in INITIALIZE.MOVE_VALUE;    --1x504
                  CEMEAN                : out AVG_VALUE;
                  NUM_STRINGS           : in INTEGER);         --1x16 float

  --Calculates the fitness of a population of bit strings based on performance versus
  --last five environments
  procedure GRADE_TWO (ALL_KE     : in OLD_ENV;
                       KGPOOL     : in INITIALIZE.BIT_STRINGS;
                       NEW_KGPOOL : out INITIALIZE.BIT_STRINGS;
                       ALL_STR_VAL: out FLOAT;
                       WENV       : in INITIALIZE.ENV_VALUE;
                       WMOVE      : in INITIALIZE.MOVE_VALUE;
                       KEMT       : out AVG_VALUE;
                       NUM_STRINGS: in INTEGER);


  --Selects the positions which are mated.  All bit positions after and
  --including the crossover site will be mated.
  procedure XVRSIT (MSITE : out INTEGER);


  --Determines the fitness of the previous decision in the current environment
  --(i.e. should we change our tactics, or stick with what we're doing.) G1,G2
  --Determines decision effectiveness over time
  procedure DECISION_FIT (DEC           : in out DECI_FIT;
                          ENV           : in out INITIALIZE.ENV_NUM;
                          WMOVE         : in out INITIALIZE.MOVE_VALUE;
                          WENV          : in out INITIALIZE.ENV_VALUE;
                          KGPOOL        : in out INITIALIZE.BIT_STRINGS;
                          T_RECORD      : in out DATA.TURN_RECORD;
                          THIS_ENV      : in out DATA.ENV_CHOICE;
                          KGTURN        : in INTEGER;
                          KE            : out INTEGER;
                          LAST_FIVE     : in out OLD_ENV;
                          NUM_STRINGS   : in INTEGER);
```

```
   --Mates the strings selected for reproduction
   procedure XVR (IC    : in out INITIALIZE.BIT_STRINGS;
                  KDAD,
                  KMOM,
                  MSITE : in INTEGER);

   --Inverts the alleles (bits at a certain position) in the bit string
   procedure INVERT (IC    : in out INITIALIZE.BIT_STRINGS;
                     NUM_STRINGS : in INTEGER);

   --Conducts the mutation of bit strings, if the probability of mutation
   --is met.  The number of bits is determined, and then they are chosen at
   --random.
   procedure MUTANT (IC     : in out INITIALIZE.BIT_STRINGS;
                     NUM_STRINGS : in INTEGER);


   --The main portion of the genetic algorithm:  produces the optimal
   --valuated state space.
   procedure MY_GENALG (KGPOOL       : in out INITIALIZE.BIT_STRINGS;
                        WENV         : in out INITIALIZE.ENV_VALUE;
                        WMOVE        : in out INITIALIZE.MOVE_VALUE;
                        FIVE_ENV     : in OLD_ENV;
                        NUM_STRINGS  : in INTEGER);


end GENALG;
```

```
--Title:      genalg_b.a
--Subject:    Contains the genetic algorithm procedures for the simulator
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

with INITIALIZE, DATA, TEXT_IO, U_RAND;
use TEXT_IO;

package body GENALG is

 package INTEGER_INOUT is new INTEGER_IO (INTEGER);
 package FLOAT_INOUT is new FLOAT_IO (FLOAT);

 use INTEGER_INOUT, FLOAT_INOUT;
 procedure GRADE(ALL_KE                 : in OLD_ENV;
                 IC                     : in out INITIALIZE.BIT_STRINGS;
                                         --NUM_STRINGSx512, value of 1-504
                 ALL_STR_VAL            : out FLOAT;  --1xNUM_STRINGS float
                 WENV                   : in INITIALIZE.ENV_VALUE;     --1x512
                 WMOVE                  : in INITIALIZE.MOVE_VALUE;    --1x504
                 CEMEAN                 : out AVG_VALUE;
                 NUM_STRINGS            : in INTEGER) is

   CESUM : FLOAT := 0.0;
   CE    : FLOAT := 0.0;
   TOTAL : AVG_VALUE;
   MAX   : INTEGER;
   ITEMP : INTEGER;
   JT    : FLOAT;
   TEMP_ALL_STR_VAL : FLOAT := 0.0;
   CEQN  : STR_VALUE;
   TEMP_IC : INITIALIZE.BIT_STRINGS;
   TEMP_CEMEAN  : AVG_VALUE;

 begin
   --Grades each bit string
   for IX in 1..NUM_STRINGS loop
     CEQN (IX) := 0.0;
     CE := 0.0;
       --For all alleles, determines the (environment - tactic) value
       for IY in 1..512 loop
         CE := (WENV (IY)) - WMOVE (IC (IX, IY)) + CE;
       end loop;   --IY

     --The value for a particular bit string, including the weight factor
     --All are equally weighted here
     --Holds the total value for the out parameter
     TEMP_ALL_STR_VAL := TEMP_ALL_STR_VAL + CE;
     --The value for this string
     CEQN (IX) := CE;
     --The sum of all bit string values
     CESUM := CEQN (IX) + CESUM;
   end loop;

   --The out parameter
   ALL_STR_VAL := TEMP_ALL_STR_VAL;

   --Rank and sort the bit strings
   for I in 1..(NUM_STRINGS-1) loop
     MAX := I;
     for J in (I+1)..NUM_STRINGS loop
       if (CEQN (J) > CEQN (MAX)) then
         for K in 1..512 loop
           ITEMP := IC (MAX, K);
```

```
          IC (MAX, K) := IC (J,K);
          IC (J, K) := ITEMP;
        end loop;
        JT := CEQN (MAX);
        CEQN (MAX) := CEQN (J);
        CEQN (J) := JT;
      end if;
    end loop;
  end loop;


  --The percent of possible value for each bit string
  TEMP_CEMEAN (1) := CEQN (1);
  for N in 2..NUM_STRINGS loop
    TEMP_CEMEAN (N) := TEMP_CEMEAN (N-1) + CEQN (N);
  end loop;

  for N in 1..NUM_STRINGS loop
    CEMEAN (N) := TEMP_CEMEAN(N) /TEMP_ALL_STR_VAL;
  end loop;

end GRADE;


--Calculates the fitness of a bit string
procedure GRADE_TWO (ALL_KE     : in OLD_ENV;
                     KGPOOL     : in INITIALIZE.BIT_STRINGS;
                     NEW_KGPOOL : out INITIALIZE.BIT_STRINGS;
                     ALL_STR_VAL: out FLOAT;
                     WENV       : in INITIALIZE.ENV_VALUE;
                     WMOVE      : in INITIALIZE.MOVE_VALUE;
                     KEMT       : out AVG_VALUE;
                     NUM_STRINGS: in INTEGER) is

  --The record holds the value of the string and it's spot among the
  --strings in the population
  type GRADE_STRING is
    record
      TOTAL_VALUE : FLOAT := 0.0;
      SPOT        : INTEGER;
    end record;

  --One record per bit string
  type STRING_SUM is array (1..NUM_STRINGS) of GRADE_STRING;
  TOTAL_GRADE      : STRING_SUM;       --The sum of all the strings values
  X,                                   --Holds the temp sum for KEMT
  TEMP_ALL_STR_VAL,                    --Holds the combined values of all strings
  SUM,                                 --Holds the value of a string
  JOINT_VAL        : FLOAT := 0.0;     --The value of a move/env combination
  ITEMP,                               --Holds a string value during sort
  MOVE_VAL,                            --Point value for a certain environment
  ENV_VAL          : FLOAT;            --Point value for a certain environment
  ITEMP_INT,                           --Holds the spot of a string during the sort
  MOVE_NUM,                            --The number of the move from the bit string
  MAX              : INTEGER;          --Used in the sort of the strings
  XXX              : FLOAT;

  type WEIGHT is array(1..5) of FLOAT;

  --Relative weights used to evaluate string fitness
  NEW_WEIGHTS : WEIGHT := (1.2, 1.1, 1.0, 0.9, 0.8);

begin
```

```
  --Each string is checked with the last five environments and the sum of those
  --five joint values is determined

  for V in 1..NUM_STRINGS loop
    for I in 1..5 loop
      --The string is checked the particular environment and the move is retrieved.
      --The point values are combined to the joint value, and then summed for the
      --last five environments.
      MOVE_NUM := KGPOOL(V, ALL_KE(I));
      MOVE_VAL := WMOVE (MOVE_NUM);
      ENV_VAL  := WENV (ALL_KE(I));
      JOINT_VAL:= (ENV_VAL - MOVE_VAL)*(NEW_WEIGHTS(I));                    --NEW
      SUM := SUM + JOINT_VAL;
    end loop;

    --This sum is entered in the record for each string
    --(-3.5) is used as an artificial floor
    SUM := SUM + 3.5;
    TOTAL_GRADE (V).TOTAL_VALUE := SUM;
    --The value of all strings put together
    TEMP_ALL_STR_VAL := TEMP_ALL_STR_VAL + SUM;
    --The initial spot of the string, before sorting
    TOTAL_GRADE (V).SPOT := V;
    SUM := 0.0;
  end loop;

  --The out parameter value
  ALL_STR_VAL := TEMP_ALL_STR_VAL;

  --Rank (sort) the bit strings
  for I in 1..(NUM_STRINGS-1) loop
    MAX := I;
    for J in (I+1)..NUM_STRINGS loop
      if (TOTAL_GRADE(J).TOTAL_VALUE > TOTAL_GRADE(MAX).TOTAL_VALUE) then
        ITEMP := TOTAL_GRADE(MAX).TOTAL_VALUE;
        ITEMP_INT := TOTAL_GRADE (MAX).SPOT;
        TOTAL_GRADE(MAX).TOTAL_VALUE := TOTAL_GRADE(J).TOTAL_VALUE;
        TOTAL_GRADE (MAX).SPOT := TOTAL_GRADE (J).SPOT;
        TOTAL_GRADE(J).TOTAL_VALUE := ITEMP;
        TOTAL_GRADE (J).SPOT := ITEMP_INT;
      end if;
    end loop;
  end loop;

  --Based on the spot in the sort, the new strings are entered in order
  for I in 1..NUM_STRINGS loop
    for K in 1..512 loop
      NEW_KGPOOL(I,K) := KGPOOL(TOTAL_GRADE(I).SPOT, K);
    end loop;
  end loop;


  --The percentage each string has of the total value of all strings
  --This is used in MY_GENALG to pick the new generation
  for I in 1..NUM_STRINGS loop
    X := TOTAL_GRADE(I).TOTAL_VALUE + X;
    KEMT(I) := X/TEMP_ALL_STR_VAL;
  end loop;

end GRADE_TWO;



--Selects the positions which are mated.  All bit positions after and
```

```
--including the crossover site will be mated.
procedure XVRSIT (MSITE : out INTEGER) is

  RM : FLOAT;
  XXX : FLOAT;
  Q   : INTEGER;                                        --used for testing

begin
  XXX := U_RAND.NEXT;
  --Checking each position until the random value is less than the RM value,
  --determines which one will be the crossover sight
  MSITE := INTEGER((XXX * 511.0) + 1.0);

end XVRSIT;


--Mates the strings selected for reproduction
procedure XVR (IC    : in out INITIALIZE.BIT_STRINGS;       --8x512 of int
               KDAD,
               KMOM,
               MSITE : in INTEGER) is

  I6,
  I9     : INTEGER;

begin
  --Swaps the values of the positions picked for mating
  for I69 in MSITE..512 loop
    I6 := IC (KDAD, I69);
    I9 := IC (KMOM, I69);
    IC (KDAD, I69) := I9;
    IC (KMOM, I69) := I6;
  end loop;
end XVR;


--Inverts(switches) the alleles (bits at a certain position) in the bit string
procedure INVERT (IC    : in out INITIALIZE.BIT_STRINGS;      --8x512 of int
                  NUM_STRINGS : in INTEGER) is

  TEMP : INTEGER;
  PINV  : FLOAT := 0.25;       --The prob of inversion
  XXX   : FLOAT;
  IT,
  IFS,                         --Used as a counter
  ISITES,
  ISITEF : INTEGER;            --The inversion sites

begin
  for NI in 1..NUM_STRINGS loop
    XXX := U_RAND.NEXT;
    --No inverting if the random number is greater the the prob of inversion
    if XXX > PINV then
      exit;
      PUT_LINE ("GENALG.INVERT  I shouldn't be here.");
    end if;

    --First position for inversion
    XXX := U_RAND.NEXT;
    ISITES := INTEGER (512.0 * XXX);
    --Last position for inversion
    XXX := U_RAND.NEXT;
    ISITEF := INTEGER (512.0 * XXX);
    --Since there is no "0" position in the string
```

```
    if ISITES = 0 then
      ISITES := 1;
    end if;

    --Since there is no "0" position in the string
    if ISITEF = 0 then
      ISITEF := 1;
    end if;

    --If the positions are identical
    if (ISITES = ISITEF) then
      if ISITEF < 512 then
        ISITEF := ISITES + 1;
      else
        exit;
      end if;
    end if;

    --If the start position is greater than the end position, swap them
    if (ISITES > ISITEF) then
      IT := ISITEF;
      ISITEF := ISITES;
      ISITES := IT;
    end if;

    --The number of alleles to be inverted
    IFS := (ISITEF - ISITES);

    --The alleles are switched
    --TEMP holds the values of the alleles to be inverted
    while IFS > 0 loop
      TEMP := IC (NI, ISITES);
      IC (NI, ISITES) := IC (NI, ISITES + IFS);
      IC (NI, ISITES + IFS) := TEMP;
      IFS := IFS - 2;
      ISITES := ISITES + 1;
    end loop;
  end loop;

end INVERT;


--Conducts the mutation of bit strings, if the probability of mutation
--is met.  The number of bits is determined, and then they are chosen at
--random.
procedure MUTANT (IC     : in out INITIALIZE.BIT_STRINGS;          --16x512 of int
                  NUM_STRINGS  : in INTEGER) is

  PM    : FLOAT := 0.001;                      --Prob of mutation
  BITS  : FLOAT := FLOAT(NUM_STRINGS)* 512.0;--Total number of alleles in the strings
  BMUT  : FLOAT := PM * BITS;                  --Total number of alleles to be mutated
  IBMUT : INTEGER;                             --BMUT converted to integer
  X1,
  X2,
  X3    : FLOAT;                               --Used to hold the random numbers
  R1    : FLOAT;                               --Holds the string to be considered
  R2    : FLOAT;                               --Holds the position to be considered
  J1,
  J2    : INTEGER;                             --The integer conversions for R1 and R2
  XM    : FLOAT;                               --The new value for the mutated allele

begin

  IBMUT := INTEGER(BMUT);
```

```
  --Looping through the number of alleles to be mutated
  for I in 1..IBMUT loop
    X1 := U_RAND.NEXT;
    X2 := U_RAND.NEXT;
    X3 := U_RAND.NEXT;
    --Pick the string for the mutations
    R1 := (X1 * FLOAT(NUM_STRINGS - 1));
    J1 := INTEGER(R1) + 1;
    --Pick the position of the allele
    R2 := (X2 * 511.0);
    J2 := INTEGER(R2) + 1;
    --Pick the new value for the allele
    XM := X3 * 503.0;
    --Enter the new value
    IC (J1, J2) := INTEGER(XM) + 1;
  end loop;

end MUTANT;



--The main portion of the genetic algorithm:  produces the optimal
--valuated state space.
procedure MY_GENALG (KGPOOL       : in out INITIALIZE.BIT_STRINGS;
                     WENV         : in out INITIALIZE.ENV_VALUE;--1x512 of float
                     WMOVE        : in out INITIALIZE.MOVE_VALUE;--1x504 of float
                     FIVE_ENV     : in OLD_ENV;
                     NUM_STRINGS  : in INTEGER) is

  type GET_MATE   is array (1..NUM_STRINGS) of BOOLEAN;
  type CROSS_SITE is array (1..(NUM_STRINGS/2)) of INTEGER;
  type MATES      is array (1..NUM_STRINGS) of INTEGER;

  NEW_IB,           --Holds the reordered offspring after mating, inversion, mutation
  NEW_KGPOOL,       --Holds the reordered parent strings
  IB,               --Holds the strings after being reordered for mating
  NEW_TEMP,         --Holds the reordered offspring strings
  TEMP     :  INITIALIZE.BIT_STRINGS;--Holds the strings selected from the
                                     --parentgeneration

  XXX      : FLOAT;

  PXVR     : FLOAT := 0.65;    --Prob of successful mating
  R,                           --Holds spot of string when picking new generation
  MSITEA   : INTEGER;          --Designates crossover site

  CROSS_SITES  : CROSS_SITE;         --The crossover sites for mating
  MATE_LIST : MATES;                 --The order of the strings for mating
  KEMT     : AVG_VALUE;              --Holds the relative values of the strings in a
                                     --population

  MATE_CHOICE  : INTEGER := 1;       --Used to designate a mate choice
  ODD_STRING,
  COUNT        : INTEGER := 1;       --Used as counters for mate choices

  STR_VAL_PARENT,                   --holds the total value of the parent strings
  STR_VAL_KIDS       : FLOAT := 0.0;--holds the total value of the offspring strings


begin

  --Number of generations of selecting the fittest arrays for mating
  for IG in 1..5 loop
    GRADE (FIVE_ENV,KGPOOL,STR_VAL_PARENT,WENV,WMOVE, KEMT, NUM_STRINGS);
```

```
--Based on the relative strengths of the parent strings(KEMT), the strings are
--chosen for the new generation
for K in 1..NUM_STRINGS loop
  XXX := U_RAND.NEXT;
  for I in 1..NUM_STRINGS loop
    if XXX < KEMT(I) then
      R := I;
      exit;
    end if;
  end loop;

  --The allele values are copied over
  for L in 1..512 loop
    TEMP (K, L) := KGPOOL (R, L);
  end loop;
end loop;

--The new strings are ranked and sorted
GRADE (FIVE_ENV,TEMP,STR_VAL_PARENT,WENV,WMOVE, KEMT, NUM_STRINGS);

--Used as counters/position holders for picking mates
ODD_STRING := 1;
COUNT := 1;

--To avoid "incest" the strings are not mated with each other.  The identical
--strings are initially adjacent after the sorting.
while ODD_STRING < NUM_STRINGS loop
  for L in 1..512 loop
    --The mates are picked, and placed adjacent to each other
    IB (ODD_STRING, L) := TEMP (COUNT, L);
    IB (ODD_STRING + 1, L) := TEMP (COUNT + (NUM_STRINGS/2), L);
  end loop;
  --Counts through the strings
  ODD_STRING := ODD_STRING + 2;
  COUNT := COUNT + 1;
end loop;

--The strings are mated (using XVR procedure) with their neighbor
MATE_CHOICE := 1;
while MATE_CHOICE < NUM_STRINGS loop
  XXX := U_RAND.NEXT;
  if XXX <= PXVR then
     XVRSIT (MSITEA);
     XVR (IB,MATE_CHOICE, MATE_CHOICE + 1,MSITEA);
   end if;
   --Increment for the next pair of mates
   MATE_CHOICE := MATE_CHOICE + 2;
 end loop;

--Inversion
INVERT (IB,  NUM_STRINGS);

--Mutation
MUTANT (IB,  NUM_STRINGS);

--Calculate the fitness of the IB arrays after inversion and mutation
GRADE_TWO(FIVE_ENV,IB,STR_VAL_PARENT,WENV,WMOVE, KEMT, NUM_STRINGS);

--Decides which generation will be chosen, based on total value
if STR_VAL_PARENT < STR_VAL_KIDS then
  for I in 1..NUM_STRINGS loop
    for J in 1..512 loop
      KGPOOL (I,J) := IB (I, J);
```

```
          end loop;
        end loop;
      end if;

    --Proceed to the next generation
    end loop;

  end MY_GENALG;


  --Determines the fitness of the previous decision in the current environment
  --(i.e. should we change our tactics, or stick with what we're doing.)
  --Determines decision effectiveness over time
  procedure DECISION_FIT (DEC           : in out DECI_FIT;
                          ENV           : in out INITIALIZE.ENV_NUM;
                          WMOVE         : in out INITIALIZE.MOVE_VALUE;
                          WENV          : in out INITIALIZE.ENV_VALUE;
                          KGPOOL        : in out INITIALIZE.BIT_STRINGS;
                          T_RECORD      : in out DATA.TURN_RECORD;
                          THIS_ENV      : in out DATA.ENV_CHOICE;
                          KGTURN        : in INTEGER;
                          KE            : out INTEGER;
                          LAST_FIVE     : in out OLD_ENV;
                          NUM_STRINGS   : in INTEGER) is

    KG    : INTEGER;          --Holds the environment number

  begin
    --The current environment number
    KG := ENV (THIS_ENV.IVC1, THIS_ENV.IVC2, THIS_ENV.IVC3, THIS_ENV.IVC4,
               THIS_ENV.IVC5, THIS_ENV.IVC6, THIS_ENV.IVC7);

    --Historical data
    T_RECORD.TENV (KGTURN) := KG;

    --Maintains queue of previous environment numbers
    for I in reverse 2..5 loop
      LAST_FIVE (I) := LAST_FIVE (I-1);
    end loop;
    LAST_FIVE (1) := KG;

    --Replaces former turns in the "queue" of decision values
    for X in 1..NUM_STRINGS loop
      for Y in reverse 1..4 loop
        DEC (X, Y+1) := DEC (X, Y);
      end loop;
    end loop;

    --Determine the new values for this state space = the most recent value
    for X in 1..NUM_STRINGS loop
      DEC (X, 1) := WMOVE (KGPOOL(X, KG)) - FLOAT(WENV (KG));
    end loop;


    --Call the genetic algorithm to determine the new values
    MY_GENALG (KGPOOL, WENV, WMOVE, LAST_FIVE, NUM_STRINGS);

    --New environment number
    KE := KG;

  end DECISION_FIT;


end GENALG;
```

```
--Title:          turn_end_s.a
--Subject:        Updates the debrief materials, advances the clock and
--                turn count.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

with DATA, INITIALIZE;

package END_TURN is

  --Updates the value in the ship and sub record
  procedure RECORD_UPDATE (SHIP_RECORD : in out DATA.SHIP_DATA;
                           SUB_RECORD  : in out DATA.SUB_DATA;
                           NEW_TURN    : in out  DATA.TURN_RECORD;
                           KGTURN      : in INTEGER;
                           WENV        : in INITIALIZE.ENV_VALUE;
                           WMOVE       : in INITIALIZE.MOVE_VALUE);

  --Updates the clock
  procedure TIME_AND_TURN (IELAPT : in out INTEGER;
                           TIMER  : in out DATA.TIME_RECORD;
                           KGTURN : in out INTEGER);

  --Plays back the data from the game
  procedure WASH_UP(LAST_TURN : in out DATA.TURN_RECORD;
                    KG_TURN   : in out INTEGER);


end END_TURN;
```

```
--Title:          turn_end_b.a
--Subject:        Updates the debrief materials, advances the clock and
--                turn count.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

with  MATH_LIB, TEXT_IO;
use MATH_LIB, TEXT_IO;

package body END_TURN is

  package FLOAT_INOUT is new FLOAT_IO (FLOAT);
  package INTEGER_INOUT is new INTEGER_IO (INTEGER);

  use FLOAT_INOUT, INTEGER_INOUT;


  --Determines the distance the ship/sub traveled in the x and y coord-
  --inates.
  procedure XYSTEP (CSE,
                    SPD  : in FLOAT;
                    XSTEP,
                    YSTEP : out FLOAT) is

    MY_PI : FLOAT := 3.14159;
    PI_12 : FLOAT := MY_PI/2.0;
    PI_32 : FLOAT := 3.0*MY_PI/2.0;
    TSTEP : FLOAT := 3.0;          --3 minute turn period
    RDCSE : FLOAT;                 --Course in radians
    DIST  : FLOAT;
    RANG  : FLOAT;                 --Normalized radian course

  begin
    --Convert course from degrees to radians
    RDCSE := CSE * MY_PI/180.0;
    --Determine distance traveled during the timestep
    DIST := SPD * (TSTEP/60.0);

    --Note that course angles are measured from a 12 o'clock position
    --clockwise
    --Convert the cse/spd to X & Y changes, based on what quadrant the
    --course falls in.
    if RDCSE <= PI_12 then
      XSTEP := MATH_LIB.SIN(RDCSE) * DIST;
      YSTEP := MATH_LIB.COS(RDCSE) * DIST;

    elsif RDCSE <= MY_PI then
      RANG := RDCSE - PI_12;
      XSTEP := MATH_LIB.COS(RANG) * DIST;
      YSTEP := -1.0 * (MATH_LIB.SIN(RANG)) * DIST;

    elsif RDCSE <= PI_32 then
      RANG := PI_32 - RDCSE;
      XSTEP := -1.0 * (MATH_LIB.COS(RANG) * DIST);
      YSTEP := -1.0 * (MATH_LIB.SIN(RANG) * DIST);
    else
      RANG := RDCSE - PI_32;
      XSTEP := -1.0 * (MATH_LIB.COS(RANG) * DIST);
      YSTEP := MATH_LIB.SIN(RANG) * DIST;
    end if;
  end XYSTEP;

  --Updates the values in the ship and sub record
  procedure RECORD_UPDATE (SHIP_RECORD : in out DATA.SHIP_DATA;
```

```
                          SUB_RECORD  : in out
                          DATA.SUB_DATA;
                          NEW_TURN    : in out DATA.TURN_RECORD;
                          KGTURN      : in INTEGER;
                          WENV        : in INITIALIZE.ENV_VALUE;
                          WMOVE       : in INITIALIZE.MOVE_VALUE) is

  XUSTEP,
  YUSTEP,
  XVSTEP,
  YVSTEP   : FLOAT;
  THIS_ENV,
  THIS_MOVE : INTEGER;

begin

  XYSTEP(SHIP_RECORD.VCSE, SHIP_RECORD.VSPD, XVSTEP, YVSTEP);
  XYSTEP(SUB_RECORD.UCSE, SUB_RECORD.USPD, XUSTEP, YUSTEP);

  SUB_RECORD.XU := SUB_RECORD.XU + XUSTEP;
  SUB_RECORD.YU := SUB_RECORD.YU + YUSTEP;
  SHIP_RECORD.XV := SHIP_RECORD.XV + XVSTEP;
  SHIP_RECORD.YV := SHIP_RECORD.YV + YVSTEP;

  --Correct the course if necessary
  if SHIP_RECORD.VCSE > 360.0 then
    SHIP_RECORD.VCSE := SHIP_RECORD.VCSE - 360.0;
  elsif SHIP_RECORD.VCSE < 0.0 then
    SHIP_RECORD.VCSE := SHIP_RECORD.VCSE + 360.0;
  end if;

  if SUB_RECORD.UCSE > 360.0 then
    SUB_RECORD.UCSE := SUB_RECORD.UCSE - 360.0;
  elsif SUB_RECORD.UCSE < 0.0 then
    SUB_RECORD.UCSE := SUB_RECORD.UCSE + 360.0;
  end if;

  --Enter into the history
  NEW_TURN.TXU(KGTURN) := SUB_RECORD.XU;
  NEW_TURN.TYU(KGTURN) := SUB_RECORD.YU;
  NEW_TURN.TXV(KGTURN) := SHIP_RECORD.XV;
  NEW_TURN.TYV(KGTURN) := SHIP_RECORD.YV;
  NEW_TURN.TZU(KGTURN) := SUB_RECORD.ZU;
  NEW_TURN.TZV(KGTURN) := SHIP_RECORD.ZV;

  NEW_TURN.TUC(KGTURN) := SUB_RECORD.UCSE;
  NEW_TURN.TUS(KGTURN) := SUB_RECORD.USPD;
  NEW_TURN.TVC(KGTURN) := SHIP_RECORD.VCSE;
  NEW_TURN.TVS(KGTURN) := SHIP_RECORD.VSPD;
  NEW_TURN.UATK(KGTURN) := SUB_RECORD.TORPAV;
  NEW_TURN.TDUCSE(KGTURN) := SUB_RECORD.DUCSE;

  --Reset the "delta" course
  SUB_RECORD.DUCSE := 0.0;

  THIS_ENV := NEW_TURN.TENV (KGTURN);
  THIS_MOVE := NEW_TURN.TMOVE (KGTURN);
  NEW_TURN.TPOINTS (KGTURN) := WENV (THIS_ENV) - WMOVE (THIS_MOVE);

end RECORD_UPDATE;

--Updates the clock
procedure TIME_AND_TURN (IELAPT : in out INTEGER;
                         TIMER  : in out DATA.TIME_RECORD;
```

```
                          KGTURN : in out INTEGER) is

   ITSTEP : INTEGER := 3;

begin
   IELAPT := IELAPT + ITSTEP;
   TIMER.GMIN := TIMER.GMIN + ITSTEP;

   --Advance the hour
   if TIMER.GMIN > 59 then
     TIMER.GMIN := TIMER.GMIN - 60;
     TIMER.GHR := TIMER.GHR + 1;
   end if;

   --Advance the day
   if TIMER.GHR > 23 then
     TIMER.GHR := TIMER.GHR - 24;
     TIMER.GDAY := TIMER.GDAY + 1;
   end if;

   KGTURN := KGTURN + 1;
   NEW_LINE(2);
end TIME_AND_TURN;


procedure WASH_UP (LAST_TURN : in out DATA.TURN_RECORD;
                   KG_TURN   : in out INTEGER) is

   X,
   W : INTEGER;
   V : POSITIVE_COUNT;
   Q,
   R : POSITIVE_COUNT;
   M : POSITIVE_COUNT := 6;
   RANGE_COUNT : INTEGER := 1;
   ATTACK_COUNT : INTEGER := 0;

begin
   X := INTEGER(KG_TURN/10);

   if LAST_TURN.UATK(1) then
     ATTACK_COUNT := 1;
   end if;

   for Y in (0..X) loop
     PUT ("TURN : ");
     for Z in 1..10 loop
       PUT (((Y*10)+Z), WIDTH => 6);
     end loop;
     NEW_LINE(2);

     PUT ("DUCSE: ");
     for Z in 1..10 loop
       PUT (INTEGER(LAST_TURN.TDUCSE((Y*10)+Z)), WIDTH => 6);
     end loop;
     NEW_LINE;

     PUT ("USPD : ");
     for Z in 1..10 loop
       PUT (INTEGER(LAST_TURN.TUS((Y*10)+Z)), WIDTH => 6);
     end loop;
     NEW_LINE;

     PUT ("ATK  :   ");
```

```
for Z in 1..10 loop
  if (LAST_TURN.UATK((Y*10)+Z)) then
    PUT ("TRUE  ");
  else
    PUT ("FLSE  ");
  end if;
end loop;
NEW_LINE;

PUT ("MOVE : ");
for Z in 1..10 loop
  PUT (LAST_TURN.TMOVE((Y*10)+Z), WIDTH => 6);
end loop;
NEW_LINE;

PUT ("DEPTH: ");
for Z in 1..10 loop
  PUT (INTEGER(LAST_TURN.TZU((Y*10)+Z)), WIDTH => 6);
end loop;
NEW_LINE;

PUT ("ENV  : ");
for Z in 1..10 loop
  PUT (LAST_TURN.TENV((Y*10)+Z), WIDTH => 6);
end loop;
NEW_LINE;

PUT ("VSPD : ");
for Z in 1..10 loop
  PUT (INTEGER(LAST_TURN.TVS((Y*10)+Z)), WIDTH => 6);
end loop;
NEW_LINE;

PUT ("VCSE : ");
for Z in 1..10 loop
  PUT (INTEGER(LAST_TURN.TVC((Y*10)+Z)), WIDTH => 6);
end loop;
NEW_LINE;

PUT ("POINTS: ");
V := 1;
R := 10;
for T in V..R loop
  Q := (5 + T*M);
  SET_COL(Q);
  W := INTEGER(T);
  PUT (LAST_TURN.TPOINTS((Y*10)+W),FORE => 1, AFT => 2, EXP => 0);
end loop;
NEW_LINE(2);

PUT_LINE ("POSITS: X/Y");
PUT ("SHIP  : ");
V := 1;
R := 10;
for T in V..R loop
  Q := (5 + T*M);
  SET_COL(Q);
  W := INTEGER(T);
  PUT (LAST_TURN.TXV((Y*10)+W),FORE => 3, AFT => 1, EXP => 0);
end loop;
NEW_LINE;

V := 1;
R := 10;
```

```
      for T in V..R loop
        Q := (5 + T*M);
        SET_COL(Q);
        W := INTEGER(T);
        PUT (LAST_TURN.TYV((Y*10)+W),FORE => 3, AFT => 1, EXP => 0);
      end loop;
      NEW_LINE(2);


      PUT ("SUB: ");
      V := 1;
      R := 10;
      for T in V..R loop
        Q := (5 + T*M);
        SET_COL(Q);
        W := INTEGER(T);
        PUT (LAST_TURN.TXU((Y*10)+W),FORE => 3, AFT => 1, EXP => 0);
      end loop;
      NEW_LINE;

      V := 1;
      R := 10;
      for T in V..R loop
        Q := (5 + T*M);
        SET_COL(Q);
        W := INTEGER(T);
        PUT (LAST_TURN.TYU((Y*10)+W),FORE => 3, AFT => 1, EXP => 0);
      end loop;
      NEW_LINE;

      PUT ("RANGE: ");
      V := 1;
      R := 10;
      for T in V..R loop
        Q := (5 + T*M);
        SET_COL(Q);
        W := INTEGER(T);
        PUT (LAST_TURN.TR((Y*10)+W),FORE => 3, AFT => 1, EXP => 0);
      end loop;
      NEW_LINE(4);
    end loop;
    NEW_LINE;

    for I in 2..KG_TURN loop
      if (LAST_TURN.TR(I)) < (LAST_TURN.TR(I - 1)) then
        RANGE_COUNT := RANGE_COUNT + 1;
      end if;

      if (LAST_TURN.UATK(I)) then
        ATTACK_COUNT := ATTACK_COUNT + 1;
      end if;
    end loop;

    PUT("ATTACK COUNT: ");
    PUT (ATTACK_COUNT, WIDTH => 5);
    NEW_LINE;
    PUT("Number of turns range decreased: ");
    PUT (RANGE_COUNT, WIDTH => 5);
    NEW_LINE;
  end WASH_UP;
end END_TURN;
```

```
--Title:        data_s.a
--Subject:      Contains data and record types
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package DATA is

 type GEN_ARRAY_TF    is array (1..2)         of BOOLEAN;
 type GEN_ARRAY_FLOAT is array (1..2)         of FLOAT;
 type GEN_ARRAY_2X2   is array (1..2, 1..2)   of FLOAT;
 type GEN_ARRAY_7     is array (1..7)         of FLOAT;
 type GEN_ARRAY_4     is array (1..4)         of FLOAT;
 type GEN_ARRAY_6     is array (1..6)         of FLOAT;
 type PROP_LOSS       is array (1..5, 1..25)  of FLOAT;
 type TURN_ARRAY      is array (1..500)       of FLOAT;
 type TURN_ARRAY_0    is array (0..500)       of FLOAT;
 type TURN_ARRAY_2    is array (1..500, 1..2) of FLOAT;
 type TURN_ARRAY_3    is array (1..500)       of INTEGER;
 type TURN_ARRAY_TF   is array (1..500)       of BOOLEAN;

 --Contains the data associated with the ship, set to initial conditions
 type SHIP_DATA is
   record
     VCSE          : float := 080.0;   --Ship's course
     VSPD          : float := 15.0;    --Ship's speed
     ZV            : float := 90.0;    --Array depth
     KH              : INTEGER;          --Sonar depth (Equivalent to ZV)
     XV            : float := 230.0;   --Ship's X coordinate
     YV            : float := 250.0;   --Ship's Y coordinate
     VCTC          : GEN_ARRAY_TF := (FALSE, FALSE);     --Ship's contact status
     VCTCBR        : GEN_ARRAY_FLOAT;  --Ship's contact's bearing
     VLCTCF        : GEN_ARRAY_TF;     --Ship's lost contact flag
     VDE           : BOOLEAN := FALSE;          --Ship's esm detection flag
     VDVR          : BOOLEAN := FALSE;          --Ship visual and radar detection
     DCHK          : BOOLEAN;          --Lost contact checker
     TH            : FLOAT;            --Angle off line of sound
     TORPAU        : BOOLEAN := FALSE; --Attack status
     CTCBRG        : FLOAT;            --contact bearing
     SHIP_RANGE    : FLOAT;            --range to contact
     VDI           : FLOAT  := 15.0;
     SETOT         : FLOAT;            --Signal excess adjusted for errors
     VDECT         : GEN_ARRAY_2X2 := (1 => (others => 0.0),2 => (others => 0.0));
     VRD           : FLOAT := -6.0;    --Recognition differential;
     VPEWD         : FLOAT := 0.92;    --prob of esm detection
     VPVIS         : FLOAT := 0.012;   --prob of visual detection
     VPRDR         : FLOAT := 0.016;   --prob of radar detection
  end record;

 --Contains data associated with the submarine, set to initial conditions
 type SUB_DATA is
   record
     UCSE          : float := 270.0;   --Sub's course
     USPD          : float := 8.0;     --Sub's speed
     ZU            : float := 60.0;    --Sub's depth
     XU            : float := 260.0;   --Sub's X coordinate
     YU            : float := 250.0;   --Sub's Y coordinate
     UCTC          : GEN_ARRAY_TF := (FALSE, FALSE);     --Sub's contact status
     UDVR          : BOOLEAN := FALSE;          --Sub visual/radar detection
     UDE           : BOOLEAN := FALSE;          --Sub detection status
     URAD          : BOOLEAN := FALSE;          --Sub radar radiating status
     PURG          : float;            --Periscope range
     PUBR          : float;            --Periscope bearing
     SUB_RANGE     : FLOAT;            --Range of target
     BTSHOT        : FLOAT;            --Bearing of torpedo shot
```

```
    BD              : float;              --Line of sound
    BT              : float;              --Current Bearing trend
    BT1             : float;              --Previous bearing trend
    BR              : float;              --Difference in last two bearing rates
    BR1             : float;              --Most recent bearing rate
    PHI             : FLOAT;              --Angle off line of sound
    DPLR            : FLOAT;              --Doppler range rate
    DPLRF           : character;          --Doppler trend
    DUCSE           : FLOAT;              --Difference in submarine course
    TORPAV          : BOOLEAN := FALSE;   --Attack status
    KD              : INTEGER;            --Sub Depth (Equivalent to ZU)
    CTCBRG          : FLOAT;              --Current contact bearing
    B2              : FLOAT;              --Previous contact bearing
    UMSLPD          : FLOAT;              --Submarine patrol speed
    LCTCT           : FLOAT;              --Last contact
    URD             : FLOAT := -2.0;      --recognition differential
    UDI             : FLOAT  := 12.0;
    SETOT           : FLOAT;              --Signal excess adjusted for errors
    UDECT           : GEN_ARRAY_2X2 := ((0.0, 0.0),(0.0,0.0));
    UCTCBR          : GEN_ARRAY_FLOAT;    --Holds contact bearings (per frequency)
    ULCTCF          : BOOLEAN;            --Contact flag
    UPVIS           : FLOAT := 0.6;       --Prob of vis detection
    UPRDR           : FLOAT := 0.85;      --Prob of Radar detection
    VRE             : CHARACTER;          --Sub range estimate;
  end record;

--Tracks the game time
type TIME_RECORD is
  record
    GDAY          : INTEGER := 8;                --Game day
    GHR           : INTEGER := 4;                --Hour
    GMIN          : INTEGER := 0;                --Minute
    MONTH         : STRING (1..3)  := "DEC";     --Month
  end record;

--Keeps all pertinent information on each turn in a record of arrays, for
--post-game analysis.  V refers to the ship, U to the sub
type TURN_RECORD is
  record
    TXU,
    TXV,
    TYU,
    TYV,          --Position info
    TZU,          --Depth of sub
    TZV,          --Depth of towed array
    TUC,
    TUS,
    TVC,
    TVS,          --Course and speed info
    TDUCSE,       --Offset from bearing towards ship
    TPOINTS       : TURN_ARRAY_0;    --Joint space value
    TENV,                            --Environment Number
    TMOVE         : TURN_ARRAY_3;    --Move number
    UATK          : TURN_ARRAY_TF;   --Attack status
    TPHI,
    TR,                              --Range between ship and sub
    TTH,
    TDEC,
    TDPLR         : TURN_ARRAY;      --Doppler status
    MVC1,
    MVC2,
    MVC3,
    MVC4,
    MVC5,
```

```
    MVC6,
    MVC7,                          --Environment values
    MUC1,
    MUC2,
    MUC3,
    MUC4        :  TURN_ARRAY_3;  --Tactics choices
    TSE         :  TURN_ARRAY_2;  --Signal excess
  end record;

--Keeps misc probabilities and ranges
type PROB_RANGE is
  record
    PESUNK      : FLOAT := 0.40;    --Probability of sinking if hit fore or aft
    PHIT        : FLOAT := 0.65;    --Probability of torpedo hit
    PMSUNK      : FLOAT := 0.80;    --Probability of sinking if hit amidships
    UTPRG       : FLOAT := 10.0;    --Sub's torpedo range (NM)
    VTPRG       : FLOAT := 2.0;     --Ship's torpedo range (NM)
  end record;

--Data used in initialization-not required after that
type SET_UP_DATA is
  record
    OCEAN       : STRING (1..5):= "WLANT";
    SUBCL       : STRING (1..6):= "VICTOR";
    SHIPCL      : STRING (1..13) := "FF1052-SQR18A";
    ROE         : STRING (1..4) := "HOT ";
    FTORPAU     : STRING (1..8) := "NOATTACK";
  end record;

--Used when determining which "environment" of inputs the sub is operating in
type ENV_CHOICE is
  record
    IVC1        : INTEGER;
    IVC2        : INTEGER;
    IVC3        : INTEGER;
    IVC4        : INTEGER;
    IVC5        : INTEGER;
    IVC6        : INTEGER;
    IVC7        : INTEGER;
  end record;
--Keeps the values for the various environmental inputs and move choices
type ENV_MOVE_VALUES is
  record
    UC1 : GEN_ARRAY_FLOAT := (9.0, 0.0);
    UC2 : GEN_ARRAY_7     := (15.0, 14.0, 10.0, 8.0, 5.0, 4.0, 3.0);
    UC3 : GEN_ARRAY_6     := (10.0, 7.0, 4.0, 4.0, 2.0, 1.0);
    UC4 : GEN_ARRAY_6     := (8.0, 9.0, 10.0, 6.0, 2.0, 1.0);
    UPSV: GEN_ARRAY_4     := (10.0, 10.0, 7.0, 3.0);
    UMW : GEN_ARRAY_4     := (1.0, 10.0, 10.0, 9.0);--Relative weight of the tactics
    VPSV: GEN_ARRAY_7     := (10.0, 10.0, 8.0, 7.0, 5.0, 7.0, 2.0);
    VMW : GEN_ARRAY_7     := (10.0, 10.0, 3.0, 2.0, 1.0, 2.0, 1.0);- Rela. wt of env
    VC1 : GEN_ARRAY_4     := (10.0, 4.0, 2.0, 0.0);
    VC2 : GEN_ARRAY_4     := (10.0, 9.0, 4.0, 0.0);
    VC3 : GEN_ARRAY_FLOAT := (3.0, 1.0);
    VC4 : GEN_ARRAY_FLOAT := (2.0, 1.0);
    VC5 : GEN_ARRAY_FLOAT := (1.0, 0.0);
    VC6 : GEN_ARRAY_FLOAT := (2.0, 1.0);
    VC7 : GEN_ARRAY_FLOAT := (1.0, 0.0);--Environment point values
  end record;
end DATA;
```

```
--Title:         env_data_s.a
--Subject:       Contains the data for ambient noise levels, prop loss figures,
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package ENV_DATA is


  type FREQ_ARRAY         is array (1..4) of FLOAT;
  type NOISE_ARRAY        is array (1..3, 1..6) of FLOAT;
  type NOISE              is array (1..7) of FLOAT;
  type PROP_LOSS_ARRAY    is array (1..5, 1..25) of FLOAT;
  type PROP_LOSS_ARRAY_2  is array (1..4, 1..5, 1..25) of FLOAT;
  type FREQ_ARRAY_2       is array (1..2) of FLOAT;

  --Keeps the decibel levels for the various sub noises
  type SUB_NOISE_RECORD is
    record
      USN         : NOISE;
      USLF3       : NOISE;
      USLF4       : NOISE;
    end record;

  --Keeps the decibel levels for the various ship noises
  type SHIP_NOISE_RECORD is
    record
      VSN         : NOISE;
      VSLF1       : NOISE;
      VSLF2       : NOISE;
    end record;

  --Keeps the prop loss levels for the target vs ship's towed body
  type PROP_LOSS is
    record
      PLF1        : PROP_LOSS_ARRAY;
      PLF2        : PROP_LOSS_ARRAY;
      PLF3        : PROP_LOSS_ARRAY_2;
      PLF4        : PROP_LOSS_ARRAY_2;
    end record;

  --The frequencies the ship and sub are searching on
  NEW_FREQS : FREQ_ARRAY := (400.0, 1200.0, 300.0, 1200.0);

  --The prop loss in decibels for combinations of target depth and speed
  --and depth of ship's towed array
  NEW_PROP_LOSS : PROP_LOSS :=
    (PLF1 => ((72.1, 79.7, 83.6, 84.9, 83.3, 82.7, 82.1, 81.7,
               81.6, 82.0, 83.2, 86.2, 93.2, 93.5, 93.5, 92.8,
               92.0, 91.3,90.8, 90.6, 90.7, 91.4, 92.7, 94.6, 97.1),
              (75.8, 81.3, 87.9,84.9, 83.5, 82.7,82.1, 81.7,
               81.7, 82.0, 83.1,86.2, 91.1, 94.1,93.4, 92.8,
               91.9,91.3,90.8,90.6,90.7,91.3,92.6,94.5,96.9),
              (77.4,87.3,87.7,85.3,84.5,83.1,82.4,81.9,
               81.9,82.7,84.8,89.2,92.5,94.3,93.8,93.0,92.3,
               91.8,91.5,91.4,91.8,92.9,94.7,96.6,98.9),
              (76.2,87.6,87.7,85.4,84.7,83.2,82.6,82.2,
               82.5,83.8,86.4,90.2,93.2,94.0,94.1,93.4,
               92.8,92.4,92.2,92.4,93.0,94.3,96.2,98.1,99.7),
              (74.7,87.6,87.6,85.7,84.8,83.3,82.6,82.3,82.7,
               84.1,86.8,90.1,93.1,94.0,94.2,93.5,92.9,92.5,
               92.4,92.6,93.3,94.6,96.5,98.4,99.7)),
     PLF2 => ((79.2,85.7,90.4,93.7,95.5,96.0,95.9,
               94.1,91.7,90.6,91.2,95.5,106.6,113.0,
               114.1,113.1,112.7,114.5,112.9,111.7,110.7,
```

```
          110.4,111.5,114.5,119.7),
         (73.7,85.3,90.4,93.5,95.4,96.0,95.8,
          93.8,91.6,90.6,91.3,95.4,103.0,112.2,
          113.7,112.9,113.1,113.8,113.1,111.5,110.6,
          110.4,111.5,114.3,119.1),
         (77.8,93.1,94.4,94.5,96.7,96.5,96.1,
          95.0,92.1,91.2,93.5,98.3,103.9,112.1,
          114.5,115.8,115.2,114.8,113.4,112.2,111.6,
          112.1,114.1,117.4,121.6),
         (76.1,93.2,95.1,95.2,96.7,96.6,96.3,
          95.1,92.6,92.4,95.5,99.4,104.6,111.8,
          115.8,116.2,115.8,115.5,114.1,113.1,112.9,
          113.8,116.0,119.1,122.8),
         (74.0,92.8,95.3,95.3,96.6,96.6,96.3,
          94.5,92.6,92.8,96.2,99.3,104.1,111.1,
          116.1,116.3,115.9,115.7,114.2,113.2,113.1,
          114.3,116.5,119.3,122.7)),
PLF3 => (1 => ((80.4,81.1,85.1,82.5,81.6,82.5,81.6,81.0,80.8,
                80.8,81.0,81.6,82.8,85.4,90.0,90.9,90.6,90.0,
                89.5,89.2,88.9,88.9,89.1,89.7,90.9),
               (73.3,82.0,85.0,82.4,81.5,81.0,80.8,80.8,81.0,
                81.5,82.5,85.3,88.9,90.8,90.5,90.0,89.5,89.1,
                88.8,88.8,88.9,89.5,90.6,92.2,94.2),
               (77.6,86.6,85.4,83.5,81.9,81.3,80.9,80.9,81.3,
                82.2,84.2,88.0,89.9,91.2,90.9,90.3,89.8,89.5,
                89.5,89.6,90.2,91.0,93.6,94.2,95.7),
               (76.6,86.9,85.5,83.7,82.0,81.4,81.1,81.3,81.9,
                83.3,85.7,88.9,90.5,91.5,91.2,90.7,90.3,90.2,
                90.2,90.6,91.4,92.3,93.9,95.4,96.5),
               (71.6,83.2,85.0,82.4,81.5,81.0,80.8,80.8,81.1,
                81.6,82.6,85.6,89.1,90.9,90.5,90.0,89.5,89.1,
                88.9,88.8,89.0,89.6,90.7,92.4,94.4)),
          2 =>((74.3,84.2,81.7,82.4,81.5,81.0,80.8,80.8,81.0,
                81.5,82.3,85.2,89.1,90.7,90.4,89.9,89.5,89.1,
                88.8,88.7,88.9,89.4,90.5,92.0,93.9),
               (76.0,86.5,85.3,83.4,82.2,81.3,80.9,80.9,81.3,
                82.2,84.3,87.2,89.8,90.9,90.8,90.2,89.8,89.5,
                89.4,89.6,90.1,91.0,92.4,94.2,95.5),
               (76.0,86.9,85.4,83.6,81.9,81.4,81.2,81.3,81.9,
                83.2,86.0,88.5,90.4,91.3,91.2,90.7,90.3,90.2,
                90.2,90.6,91.4,92.3,93.8,95.3,96.3),
               (77.3,86.7,85.4,83.5,81.8,81.3,80.9,80.9,81.3,
                82.3,84.3,87.9,89.9,91.2,90.9,90.3,89.8,89.6,
                89.5,89.7,90.2,91.2,92.7,94.4,95.8),
               (76.0,86.5,85.3,83.4,82.2,81.3,80.9,80.9,81.3,
                82.2,84.3,87.2,89.8,90.9,90.8,90.2,89.8,89.5,
                89.4,89.6,90.1,91.0,92.4,94.2,95.4)),
          3 => ((71.2,85.5,85.1,83.2,82.4,81.9,81.8,81.9,82.2,
                81.4,79.6,84.2,91.1,91.4,91.2,90.6,90.2,89.9,
                89.9,90.4,91.0,88.2,86.7,93.7,96.4),
               (71.7,85.7,85.6,86.9,82.5,82.1,82.1,82.3,82.2,
                82.7,83.3,85.9,91.4,90.1,91.5,91.0,90.7,90.6,
                90.6,91.3,92.3,89.7,89.7,89.6,97.4),
               (77.0,87.0,85.5,83.7,82.0,81.4,81.2,81.3,81.9,
                83.3,85.9,88.8,90.6,91.5,91.2,90.7,90.3,90.2,
                90.3,90.7,91.4,92.4,94.0,95.5,96.5),
               (76.1,86.9,85.4,83.7,81.9,81.4,81.2,81.3,82.0,
                83.3,86.1,88.4,90.4,91.3,91.2,90.7,90.3,90.2,
                90.2,90.6,91.4,92.3,93.8,95.3,96.3),
               (71.1,85.7,85.6,83.2,82.5,82.1,82.1,82.3,82.8,
                82.7,83.3,85.9,91.4,91.8,91.5,91.0,90.7,90.6,
                90.8,91.3,92.3,89.7,89.7,94.7,97.1)),
          4 => ((68.1,85.4,85.1,83.4,82.6,82.3,82.4,82.7,83.4,
                84.3,84.2,80.3,83.0,89.7,91.6,91.4,91.2,91.2,
```

```
                 91.6,92.3,93.4,92.4,89.8,89.7,87.4),
                (75.2,87.4,85.6,83.8,82.0,81.4,81.2,81.4,82.2,
                 83.6,86.5,88.8,90.5,91.5,91.3,90.8,90.5,90.3,
                 90.5,90.9,91.7,92.7,94.2,95.7,96.6),
                (75.3,85.7,85.5,83.0,82.0,81.4,81.2,81.4,82.2,
                 83.5,86.3,88.8,90.3,91.3,91.3,90.7,90.4,90.3,
                 90.4,90.8,91.5,92.6,94.0,95.5,96.4),
                (71.1,85.4,85.6,83.3,82.5,82.2,82.2,82.4,82.9,
                 83.0,84.5,85.9,91.0,91.8,91.5,91.1,90.8,90.7,
                 90.9,91.6,92.5,89.8,90.7,94.3,97.1),
                (68.5,83.8,85.6,83.4,82.7,82.4,82.5,82.8,83.0,
                 84.8,85.0,82.7,82.9,89.3,91.5,91.5,91.3,91.3,
                 91.7,82.5,93.4,93.0,90.7,89.8,89.8))),
PLF4 => (1 => ((73.4,80.0,83.6,86.3,88.2,89.5,90.4,90.4,89.7,
                 89.3,90.0,93.3,96.9,98.2,98.9,99.7,100.3,101.1,
                 101.7,102.2,102.6,103.2,103.9,104.9,105.9),
                (76.8,85.2,90.1,93.4,95.4,95.9,95.8,93.8,91.6,
                 90.6,91.3,95.6,103.1,112.2,113.7,112.9,113.2,
                 113.9,113.1,111.5,110.6,110.4,111.6,114.4,119.3),
                (77.7,92.9,94.4,94.7,96.7,96.5,96.0,94.2,92.1,
                 91.3,93.5,98.3,103.9,111.7,114.7,115.7,115.2,
                 114.8,113.0,112.2,111.6,112.1,114.2,117.4,121.7),
                (76.1,93.2,95.2,95.3,96.6,96.6,96.3,95.1,92.6,
                 92.5,95.5,99.5,104.7,112.0,115.7,116.2,115.8,
                 115.6,114.2,113.1,112.9,113.8,116.1,119.2,122.9),
                (75.5,85.1,90.0,93.4,95.4,95.9,95.8,93.7,91.6,
                 90.8,91.5,95.9,103.4,112.5,113.8,113.1,113.4,
                 113.4,112.6,111.6,110.7,110.6,111.7,114.6,119.6)),
         2 => ((73.0,81.3,89.7,93.2,95.3,95.9,95.0,93.1,91.5,90.7,
                 91.4,95.4,102.7,111.8,113.0,112.8,112.9,113.3,
                 112.4,111.4,110.5,110.4,111.5,114.2,118.7),
                (74.8,91.9,94.2,95.4,96.1,96.5,96.0,94.1,91.9,
                 91.4,93.6,97.9,103.5,109.9,114.6,115.6,115.1,
                 114.7,112.8,112.0,111.6,112.1,114.1,117.3,121.3),
                (74.3,92.8,94.9,95.5,96.3,96.6,96.3,94.4,92.5,
                 92.7,95.6,99.4,104.3,110.6,115.4,116.1,115.8,
                 114.7,113.6,112.9,112.9,113.8,116.0,119.0,122.7),
                (95.0,94.6,92.8,76.9,96.6,92.1,94.3,96.1,96.5,
                 91.4,111.5,104.0,98.4,93.7,114.9,113.0,114.8,
                 115.3,115.8,112.2,117.6,114.3,112.3,111.7,121.7),
                (74.8,91.9,94.2,95.4,96.1,96.5,96.1,94.1,91.9,
                 91.5,93.6,97.9,103.5,109.9,114.6,115.6,115.2,
                 114.8,112.9,112.0,111.6,112.2,114.1,117.3,121.3)),
         3 => ((74.2,89.4,95.6,96.4,96.6,96.4,96.1,94.6,93.4,
                 91.9,88.7,97.5,105.8,111.1,115.4,115.8,115.4,
                 115.5,113.8,112.8,112.8,110.6,104.4,119.7,123.7),
                (73.6,88.9,95.7,95.6,96.8,96.6,96.3,95.4,96.2,
                 92.4,91.3,88.3,106.8,105.5,115.8,116.3,116.0,
                 116.1,116.1,113.8,114.1,112.5,112.0,110.4,125.1),
                (76.0,93.3,95.2,95.4,96.5,96.7,96.3,94.6,92.6,
                 92.6,95.7,99.7,104.9,112.1,115.6,116.3,115.9,
                 115.6,114.2,113.1,113.0,114.0,116.3,119.4,123.1),
                (74.4,92.8,94.9,95.5,96.3,96.6,96.3,94.4,92.6,
                 92.7,95.6,99.4,104.3,110.6,115.4,116.1,115.8,
                 114.8,113.6,113.0,112.9,113.8,116.0,119.0,122.7),
                (73.6,88.9,95.8,96.5,96.8,96.6,96.4,95.5,94.0,
                 92.4,91.3,88.3,106.8,111.8,115.8,116.3,116.0,
                 116.1,114.4,113.8,114.1,112.5,112.0,119.7,125.0)),
         4 => ((69.5,88.5,95.9,96.5,96.9,96.8,96.6,95.4,94.7,
                 94.8,94.9,90.5,88.3,110.3,116.0,116.8,116.6,
                 116.8,115.2,114.8,115.6,115.4,110.7,109.8,112.4),
                (73.9,93.2,95.3,95.5,96.4,96.7,96.4,94.9,92.6,
                 93.0,96.4,99.6,104.4,111.3,115.9,116.3,116.0,
                 115.0,114.2,113.3,113.3,114.5,116.7,119.6,122.9),
```

```
                    (73.2,89.8,95.0,95.5,96.3,96.6,96.3,94.3,92.6,
                     93.1,96.1,99.5,104.0,110.3,115.5,116.1,115.9,
                     114.9,113.7,113.1,113.2,114.3,116.4,119.2,122.6),
                    (69.1,87.6,95.8,96.5,96.8,96.7,96.4,95.5,94.2,
                     93.2,90.2,95.6,106.4,111.4,115.5,116.4,116.1,
                     116.2,114.9,114.0,114.5,111.7,112.5,117.2,125.0),
                    (67.1,87.4,95.9,96.5,96.9,96.9,96.7,95.9,94.8,
                     95.4,94.6,91.2,92.9,100.5,115.3,116.9,116.7,
                     116.1,115.3,115.3,115.9,115.0,113.0,111.3,113.5))));


  --Contains flow noise for ship
  NEW_SHIP_NOISE : SHIP_NOISE_RECORD :=
              (VSN   => (70.0,70.5,74.0,76.0,78.5,84.0,90.0),
               VSLF1 => (140.0,143.0,146.0,150.0,155.0,162.0,170.0),
               VSLF2 => (125.0,128.0,131.0,135.0,140.0,147.0,155.0));

  --Ambient noise levels in decibels
  AMBIENT_NOISE : NOISE_ARRAY := ((69.6,69.1,69.1,68.8,68.4,68.3),
                                  (66.4,66.4,66.4,66.0,65.8,65.7),
                                  (62.3,62.3,62.3,62.3,62.3,62.3));


  --Sub flow noise in decibels
  NEW_SUB_NOISE : SUB_NOISE_RECORD :=
  (USN   => (55.0,56.0,57.5,59.5,64.0,73.0,85.0),
   USLF3 => (135.0,138.0,141.0,145.0,150.0,157.0,165.0),
   USLF4 => (120.0,123.0,126.0,130.0,135.0,142.0,150.0));


end ENV_DATA;
```

LIST OF REFERENCES

[ESH91] Eshelman, L.J., and Schaffer, J.D., "Preventing Premature Convergence in Genetic Algorithms by Preventing Incest," *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufman Publishers, Inc., 1991.

[FOG86] Fogel, L.J., Owens, A.J., and Walsh, M.J., *Artificial Intelligence through Simulated Evolution, John Wiley and Sons*, 1986

[GEN87] *Genetic Algorithms and Their Applications: International Conference on Genetic Algorithms*, L. Erlbaum Associates, 1987.

[GOL89] Goldberg, David E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[GRE86] Greffenstette, J.J., "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, 16, Number 1, 1986, The Institute of Electrical and Electronics Engineers, Inc.

[HAY91] Hayden, Timothy M., *Simulation of an Intelligent Submarine Adversary*, M.S. Thesis, Naval Postgraduate School, 1991.

[KUO 93] Kuo, T., and Hwang, S., "A Genetic Algorithm with Disruptive Selection," *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufman Publishers, Inc., 1993.

[SCH86] Schaffer, J.D., Caruana, R.A., Eshelman, L.J., Das, R., "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization," *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufman Publishers, Inc., 1989

[TAT93] Tate, D.M., and Smith,A.E., "Expected Allele Coverage and the Role of Mutation in Genetic Algorithms," *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufman Publishers, Inc., 1993.

[WHI89] Whitley, D.,"The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best," *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufman Publishers, Inc., 1989

[YAZ86] Yazdani, M. *Artificial Intelligence: Principles and Applications*, Chapman and Hall, 1986.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145

Dudley Knox Library 2
Code 052
Naval Postgraduate School
Monterey, CA 93943-5002

Dr. Ted Lewis 1
Professor and Chairman
Computer Science Department Code
Naval Postgraduate School
Monterey, CA 93943-5118

Professor J.N. Eagle, II, Code OS/Er 1
Naval Postgraduate School
Monterey, CA 93943-5118

Commander 1
Naval Ocean Systems Center
Attn: Dennis Mattison, Code 624
San Diego, CA 92152-5000

Commander 1
Naval Training Systems Center
Attn: Bob Ahlers, Code 261
12350 Research Parkway
Orlando, FL 32826

Dr. Man-tak Shing 2
Computer Science Department Code CS/Sh
Naval Postgraduate School
Monterey, CA 93943-5118

Dr. Yuh-jeng Lee 1
Computer Science Department Code CS/Le
Naval Postgraduate School
Monterey, CA 93943-5118

LCDR Michael Timmerman 2
Helicopter Antisubmarine Squadron Eight-Five
Naval Air Station North Island, CA 92135

Navy Center for Applied Research in Artificial Intelligence 1
Attn: Code 5510-GREF
4555 Overlook Ave, SW
Washington, D.C. 20375-5000